AD-A196 289

UNCLASSIFIED

DTIC FILE COPY

1

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER AFIT/CI/NR 88-161	2. GOVT ACCESSION NO.	3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle) REAL TIME CONFLICT RESOLUTION IN AUTOMATED GUIDED VEHICLE SCHEDULING		5. TYPE OF REPORT & PERIOD COVERED PHD ~~MS~~ THESIS
		6. PERFORMING ORG. REPORT NUMBER
7. AUTHOR(s) STEPHEN CRAIG DANIELS		8. CONTRACT OR GRANT NUMBER(s)
9. PERFORMING ORGANIZATION NAME AND ADDRESS AFIT STUDENT AT: PENNSYLVANIA STATE UNIVERSITY		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS
11. CONTROLLING OFFICE NAME AND ADDRESS		12. REPORT DATE 1988
		13. NUMBER OF PAGES 100
14. MONITORING AGENCY NAME & ADDRESS(if different from Controlling Office) AFIT/NR Wright-Patterson AFB OH 45433-6583		15. SECURITY CLASS. (of this report) UNCLASSIFIED
		15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)

DISTRIBUTED UNLIMITED: APPROVED FOR PUBLIC RELEASE

DTIC ELECTE AUG 0 3 1988 S D

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

SAME AS REPORT

CD

18. SUPPLEMENTARY NOTES

Approved for Public Release: IAW AFR 190-1
LYNN E. WOLAVER [illegible] 22 July 88
Dean for Research and Professional Development
Air Force Institute of Technology
Wright-Patterson AFB OH 45433-6583

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

ATTACHED

88

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

ABSTRACT

Automated Guided Vehicle Systems (AGVS) are one of the most exciting and dynamic areas in material handling research today. While there have been significant advances in material handling techniques over the past decade, the primary growth in this technology has been in the areas of robotics, automated guided vehicle (AGV) systems, and automated storage and retrieval systems (AS/RS). [illegible]

The control of automated guided vehicles (AGVs) and the development of efficient algorithms to manage material handling problems have been areas of research recently demanding much attention. ~~In the past,~~ previous researchers ~~have only~~ addressed the control of AGVs over networks containing uni-directional paths. The idea of controlling AGVs through networks containing bi-directional paths is ~~very~~ complex and considered too costly for implementation.

In this work, a shortest path algorithm is embedded within a branch-and-bound procedure to generate conflict-free AGV routes through bi-directional networks. The resulting methodology efficiently resolves conflicts that would normally occur among AGVs traveling in complex AGV systems. ~~The~~ development of this methodology is discussed in detail and the computational performance of a most encouraging representative algorithm is presented. [illegible]

The Pennsylvania State University

The Graduate School

Department of Industrial Engineering

REAL TIME CONFLICT RESOLUTION IN AUTOMATED GUIDED VEHICLE SCHEDULING

A Thesis in

Industrial Engineering and Operations Research

by

Stephen Craig Daniels

Submitted in Partial Fulfillment
of the Requirements
for the Degree of

Doctor of Philosophy

May 1988

We approve the thesis of Stephen Craig Daniels.

	Date of Signature:
[illegible] Claude D. Pegden Associate Professor of Industrial Engineering Chairman of Committee	3/11/88
[illegible] Tom M. Cavalier Assistant Professor of Industrial Engineering Thesis Advisor	3/11/88
[illegible] Pius J. Egbelu Assistant Professor of Industrial Engineering	3/11/88
[illegible] Patrick Lee Assistant Professor of Management Science	3/11/88
[illegible] Allen L. Soyster Professor of Industrial Engineering Head of Department of Industrial and Management Systems Engineering	3/11/88

ABSTRACT

Automated Guided Vehicle Systems (AGVS) are one of the most exciting and dynamic areas in material handling research today. While there have been significant advances in material handling techniques over the past decade, the primary growth in this technology has been in the areas of robotics, automated guided vehicle (AGV) systems, and automated storage and retrieval systems (AS/RS).

The control of automated guided vehicles (AGVs) and the development of efficient algorithms to manage material handling problems have been areas of research recently demanding much attention. In the past, researchers have only addressed the control of AGVs over networks containing uni-directional paths. The idea of controlling AGVs through networks containing bi-directional paths is very complex and considered too costly for implementation.

In this work, a shortest path algorithm is embedded within a branch-and-bound procedure to generate conflict-free AGV routes through bi-directional networks. The resulting methodology efficiently resolves conflicts that would normally occur among AGVs traveling in complex AGV systems. The development of this methodology is discussed in detail and the computational performance of a most encouraging representative algorithm is presented.

TABLE OF CONTENTS

TABLE OF CONTENTS (continued)

TABLE OF CONTENTS (continued)

LIST OF FIGURES

LIST OF TABLES

ACKNOWLEDGEMENTS

The author would like to acknowledge the advice, consideration, and assistance of the members of his committee. Special thanks are due Dr. C. Dennis Pegden, who motivated this study with one provocative question: Why not? His continued assistance when additional obstacles were encountered was instrumental in the completion of the final product. Grateiul appreciation is also due all the other committee members. Dr. Pius Egbelu provided critical input concerning bi-directional issues. Dr. Cavalier was especially helpful in the final review of this dissertation and provided assistance in the algorithm's final form. Dr. Patrick Lee was very helpful in providing clear and concise proofs. He was always willing to provide frank evaluation of any idea presented and suggested useful presentation approaches.

The encouragement of the author's wife, Margie, kept the spark of interest alive even in the most trying times, and was invaluable in this project's final completion. While not often expressed her help was most graciously appreciated.

Chapter 1

INTRODUCTION

Background of Study

Automated Guided Vehicles

The invention of the automated guided vehicle (AGV) dates back to the 1950's when automated guided vehicles were called driverless systems. Originally used for automated distribution centers, they are now being used in manufacturing and assembly facilities. A variety of advanced technologies have emerged to expand the computer's ability to control and coordinate AGVs. With better control the automated factory became a reality and provided numerous profitable applications for the driverless vehicle.

Over the years the evolution of new electronics technology has led to advancements in the driverless vehicle. The vacuum tube was replaced by the transistor which, after several years, was superseded by the integrated circuit. Technology marched on and the integrated circuit was eventually supplanted by the microprocessor. All these technological innovations combined to make automated guided vehicle systems possible.

An automated guided vehicle system (AGVS) is an automated material handling system which is controlled by a computer system and contains independently addressable driverless vehicles on a predefined transportation network [50]. The AGVS Product Section of the Material Handling Institute defines an AGV as:

> A vehicle equipped with automatic guidance equipment, either electromagnetic or optical. Such a vehicle is capable of following prescribed guide paths and may be equipped for vehicle programming and stop selection, blocking, and any other special functions required by the system. [13]

Technological developments may have given AGVS more flexibility and capability, but market acceptance has really given the AGVS the application variety needed to expand into the material handling applications demanded today [63]. Originally used for automated distribution centers, AGVs are now being used in nearly all large scale manufacturing and assembly facilities. These vehicles have reached the point in their technological development where they are capable of being an all-encompassing link to all warehousing systems [57].

This explosion in the industrial application of AGV systems seems to have no end in sight [27]. Industry appears to be in love with the AGV as they race to apply this technology. It seems industry managers are convinced the AGV will solve any material handling problem they have and deliver all of the benefits proponents of AGVS allege. The AGVS seems to have become the most popular way to transport materials through warehouses and factories and has made AGVs the fastest growing classes of equipment in the material handling industry [53]. All available signs show that the AGVS boom will continue unabated in the future [54].

Benefits of AGVS

In almost any material handling application, the potential benefits of an AGVS are enormous. The Material Handling Institute [13] provides the following list of benefits of AGVS:

o Economic Justification - In more and more material handling situations, AGVS are proving themselves as the most economical method of moving material. Most AGVS can be justified economically in three years or less.

o Automatic Interface with Other Systems - AGVS are designed to interface automatically with other material handling systems includeing conveyors, automatic storage/retrieval systems, production lines and other devices including elevators, doors, draw bridges, robots, shrink, and stretch wrap tunnels.

- System Accountability - Computer control allows material tracking between stations and delivery confirmation in real time. The benefits include planned delivery, transaction audit records, on-line interface to production and inventory control systems, and management information on vehicle and work station production.

- Reduced Labor and Increased Productivity - In cases where driverless vehicles are used, a substantial savings is incurred due to labor reduction. Order picking vehicles require operators to perform the picking and stowing functions; however, a substantial increase in productivity is realized since operators no longer have to perform complex paperwork functions for the material tracking process.

- Additional Reductions in Labor - In addition to the direct, material handling labor savings, there are often indirect labor reductions, including expeditors, clerks, and dispatchers.

- Additional Flexibility - As material movement needs change or plant size increases, AGVS can be expanded or modified quickly and at low cost. As requirements increase, load movement capacity in the AGVS is easily modified by adding one or more vehicles.

- Unobstructed Aisles - AGVS are installed in the floor and leave no above floor obstructions. The aisles are free for other uses as necessary. Indirectly, AGVS contribute to good housekeeping by requiring the aisles be kept free to allow vehicle traffic.

- Less Product and Equipment Damage - Studies prove there is less product and plant equipment damage when AGVS are used to move material. This is because the vehicles travel on a predetermined route and thus can not collide with racks or other obstacles.

- Reliable System Capacity - Because AGVS are composed of a number of vehicles, when one vehicle requires maintenance, productivity of the other vehicles is not affected and a high degree of system availability is maintained. All automatic guided vehicles available are equipped to allow manual override for special material or vehicle movement situations.

- Energy and Environment Conservation - AGVS require very little energy to operate, are not noisy, and leave the manufacturing or warehouse floor virtually unchanged.

As listed above, AGVS users can achieve higher utilization of manufacturing or assembly equipment through automatic dispatch of vehicles and tracking of materials throughout the facility. This automatic interface with other systems can provide the management decision makers with real-time information concerning the production process. This concept gives the manager a better sense of what is happening and better ability to react to changing market conditions.

Like other automated systems, the operation of an AGV system requires close monitoring of the movement of the vehicles to ensure orderly flow and operations. The continuing growth and acceptance of AGV technology suggests further integration of guided vehicles in flexible manufacturing systems and emphasizes the development of improved control algorithms for the routing of these vehicles.

AGV Route Control Technologies

Lord Kelvin is attributed with saying "In order to control anything, first you must be able to measure it [41]." Confusion has arisen over what this means in a material handling environment. Management seems unclear as to what a material tracking system is and what the automated machines control system is. This confusion stems from the fact that control systems often include position identification submodules as a part of their control mechanism, similar to tracking submodules.

In manufacturing, material tracking systems and control systems can be compared to sports:

> Control systems are the coaches who direct actions during the game and tracking systems are the sportscasters who present the results after the game is over. [41]

Relating this back to the material handling environment and in particular an AGV system, it becomes apparent that many control systems really only operate as tracking systems and not as control systems. Their systems are simple conveyor systems with the AGVs following the leader from origin to destination, just as a shipped box would travel along a conveyor belt from point A to point B. No real control was exercised over the box, as if it really had a choice of routes to travel or conflicts to avoid. Most AGV systems have one path from node A to node B and hence very little flexibility in routing choice is available.

As for real control systems, there are two primary AGV control configurations which are in use today [69]. Primary vehicle control may reside either in each vehicle or in a central computer.

With intelligent vehicles and a simple central controller, several features are available. On board vehicle programming or dispatching is possible. The vehicle has full knowledge of routes and blocking (traffic control among vehicles) and only requires the entry of a destination and possibly a function to perform when it gets there.

A control system using intelligent vehicles and a central control computer maintains continuous communication with the vehicle, either by radio or directly through the guidewire. This continuous communication provides the capability of full real-time monitoring of the status and position of all vehicles. Usually, in the event of a failure of the communications link to a host computer the system has multiple levels of degraded operation with the lowest residing in each individual vehicle. This type of control system is typically available for installations using up to 30 vehicles.

Control systems which use a powerful centrai computer generally have vehicles with minimal intelligence. All functions, routing and blocking, are controlled by the central computer. These systems are normally used for installations with 50-200 vehicles. Central computer communication with vehicles is typically performed through grids at discrete locations in the floor. Data concentrators in the form of mini-computers buffer all of these communication grids to allow real-time processing of data by the central vehicle controller.

These minimally intelligent AGVs usually do not contain a resident map of the routes and at each grid point receive instruction on which path to take and when to stop or proceed. In the event of a central vehicle control computer failure, degraded modes of operation (other than full manual operation) are typically not available. Vehicle malfunctions between communications points cannot be reported. Instead, notification is usually the activation of an alarm if a vehicle does not reach its next communication grid within a prescribed time limit.

The new methodology presented in this work requires a central control mechanism but does not require individual vehicle intelligence. The new control methodology should provide increased productivity, throughput, and flexibility. The overall cost of operations should be reduced through a lower requirement for the number of AGVs necessary to complete the job and a decrease in space utilized for track.

Blocking Systems

To avoid train collisions, railroads have used red, yellow, and green lights, located at approximately one mile intervals, as blocking signals. As a leading train passes a blocking signal, the signal will turn from green to red, indicating to a trailing train that a leading train occupies the next track segment. Protocol requires a trailing train to stop at a red signal, sound its whistle and if it proceeds, to do so at creep speed. As the leading train passes the next block signal the first block signal turns from red to yellow. A trailing train must stop at a yellow signal, but can then proceed with caution. As the leading train passes the second block signal forward, the original signal turns from yellow to green. Trailing trains can always pass through green signals [31].

While this blocking scheme is supposed to avoid train collisions, it would seriously restrict throughput in an AGV system. However, some method of traffic control in an AGVS network is critical as vehicle separation and collision prevention must be assured. Vehicle blocking or collision avoidance within a network is currently accomplished using one or more of the following techniques. The two most widely used are computer blocking and point-to-point blocking [56].

Computer blocking is typical of systems with smart vehicles with or without a host computer. The computer, either central or on the vehicle, knows its position on the guidepath by zone and also is aware of the presence of other vehicles in adjacent zones. If the zone ahead is clear, the vehicle may proceed, if not, the vehicle will stop. Look-ahead techniques allow bypass route selection and advance slowdown to minimize continuous stopping and starting. Look-ahead capabilities usually are not available as uni-directional paths provide few alternate routes and render the look-ahead function unnecessary. When conflicts over routes at major or critical intersections occur the user must resolve the problems by specifying vehicle or routing priorities.

As a rule, point-to-point blocking techniques are hard-wired into the facility floor and, depending on the complexity of the system, may require considerable support electronics. Point-to-point techniques employ set-reset logic elements to control path zones. Vehicles are prevented from moving by hold beacons triggered by vehicles leaving a zone downstream on the path. These hold beacons ensure there will always be an empty zone between zones containing vehicles. This technique is generally found on older systems or systems having extremely simple controls, routes, and vehicles. This technique is also referred to as a zone-blocking control system and the technique most often used when creating and testing a system using simulation.

Alternate forms of blocking used less frequently involve optics, ultrasonics, and bumpers. Optical blocking systems use an infrared transmitter and receiver on the front of each vehicle. The vehicle stops when an optical target on the rear of a leading vehicle is seen. Ultrasonic applications are similar to optical blocking except they use a sonar pulse and measure the time between the reflection of the signal. Bumper blocking systems are used by small, lightweight, assembly AGVs. They are used in situations where many slow moving vehicles are configured to physically touch the preceding vehicle. The vehicle speed is such that they will stop in the collapse range of the bumper mechanism.

The new methodology presented in this work will interface with any of the vehicles, as long as the vehicle can communicate with a host controlling computer.

Purpose of Study

The typical AGV system operating in manufacturing facilities today utilizes uni-directional guide-path systems even though Egbelu and Tanchoco [24] have shown a bi-directional flow systems may be more appropriate. The justification for the use of uni-directional guide-path system instead of bi-directional system resides primarily in the simplicity of the system design and the ease of vehicle control. The objective of this study is to develop a control procedure for routing vehicles through a bi-directional network.

Bi-directional Flow AGV Systems

The use of bi-directional guide-path systems represent an area of significant potential for AGVS application. The advantages of employing bi-directional traffic flow systems versus uni-directional systems have been well documented in systems involving highways, streets, and railroads [1]. City planners have consistently understood the need to designate streets as one-way or two-way depending on the expected traffic volume, system throughput, safety, and other economic and political reasons [67,79]. Railroad planners have selected both single and double track segments for improved system performance [31]. While there are obvious differences in operational and environmental requirements for streets and railroads versus AGVs, these applications demonstrate the importance of investigating the potential benefits bi-directional AGV systems offer over the current standard uni-directional flow systems.

Egbelu and Tanchoco [24] compared facilities production potentials while operating in a uni-directional network versus one containing bi-directional paths. Using simulation techniques, they demonstrated that the use of bi-directional networks dominated uni-directional networks in every performance measure they calculated. For a fixed number of vehicle , shop throughput increased and nonproductive vehicle travel (travel while empty) time decreased. The use of bi-directional flow networks was shown to provide increased productivity in many AGV system installations, especially those requiring only a few vehicles. For a fixed workload, significantly fewer vehicles were required to achieve a certain transport capacity in bi-directional systems than in uni-directional systems.

Also identified in the study were several factors to consider when converting network segments from uni-directional to bi-directional operation.

1. The traffic intensity through the segment.
2. The traffic control requirements.
3. The traffic load that would be imposed on other network segments, from converting from bi-directional to uni-directional and vice versa.
4. Savings or increase in total travel distance over a period of time by designating segments one way or two way.
5. Future flow requirements and possible expansion or control of the network.

Spinelli [77], in an independent follow up study, found substantial time savings and increases in AGV utilization were achieved using a bi-directional AGVS.

A commonly given reason for not adopting bi-directional flow networks by vendors is the difficulty of control and the cost of developing such control systems. Young stated

> the control problems involved with dealing with bi-directional guide-path systems are enormous. If there are more than a small handful of vehicles involved in the system, the problem becomes totally unmanageable. Guide-path networks are like belt lines around a city with various entrance and exit ramps allowing access to all freeway locations or in the case of AGVs, work stations, sidings, etc. [81]

He appears to be illustrating the inability of industry to route a vehicle against traffic without disrupting the entire AGVS flow system. The key question is: Can an effective control methodology be developed without disrupting traffic flow and without costing industry too much? Economic factors can partially justify modification expense, as there are increased profits generated from expanded productivity. A larger saving can be realized from a reduction in the required number of AGVs to perform the job. Additional savings could be realized from decreased cost in the installation of a new system since the size of the bi-directional network has been shown to be smaller than the uni-directional network.

While existing simulation data provides a strong case for bi-directional flow network systems, vehicle control creates some especially difficult problems. Young [81] called these problems "unmanageable." The most obvious problem concerns the development of "sophisticated software" to route the AGV fleet and resolve resulting conflicts among the vehicles.

The AGVS Network Model

In modeling the flow of vehicles in the AGVS network, this work makes the following assumptions:

1. All vehicles travel at the same speed.
2. All nodes have facilities for buffering or holding blocked vehicles. This implies that vehicles may pass or cross only at nodes.
3. All arcs have capacity of only one AGV at any single point.
4. A vehicle travels a distance greater than or equal to its length in one time unit. Restated this means the minimum safe separation between vehicles is one time unit.

The method each individual network provides for the buffering of vehicles and the technique used for vehicle passage within the network are two topics not addressed in this work. Those problems have been effectively addressed by Egbelu and Tanchoco [22,24].

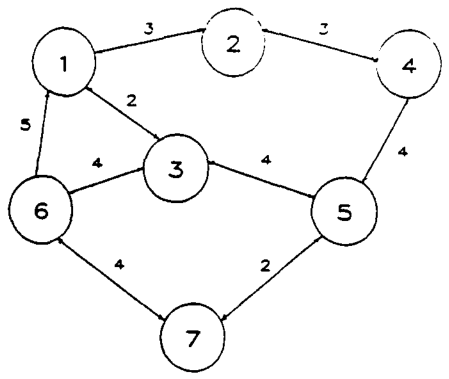
Bi-directional AGV systems present highly challenging traffic control problems. There is very little published literature on bi-directional network operations besides the classic work by Egbelu and Tanchoco [24]. Spinelli [77] considered the "only available tool for analyzing complex material handling problems," simulation, and evaluated zone segmentation lengths in bi-directional and uni-directional AGVS operation. Bi-directional control was handled using the zone-blocking method and found it only marginally effective for bi-directional networks. He found zone-blocking contained drawbacks for uni-directional systems as well.

Spinelli [77] examined the effect zone-control methodology had on AGVS performance. He found the effects of zone specification on AGVS performance were quantifiable and an important issue to consider. Extensive traffic segmenting allowed for more simultaneous occupation of track sections and more accurately modelled how AGVS operate. Failure to adequately segment the track resulted in deteriorated performance when zone-blocking control was applied. His work demonstrated the inadequacy of simulation and zone-blocking in efficiently controlling AGVS.

This work deals with the routing of vehicles over a network where some (or all) of the paths allow bi-directional flow. Any network which contains at least one bi-directional arc will be called a bi-directional network. To get an idea of the difficulty this routing problem presents, consider the following small example presented in Figure 1. There are only seven nodes with nine bi-directional paths in this network. The problem requires the scheduling of three AGVs over their respective route such that no collisions occur. It should be fairly clear that an optimal scheduling of this problem's three AGVs is a non-trivial task.

Network Information

Node	Node	Cost
1	2	3
1	3	2
1	6	5
2	4	3
3	5	4
3	6	4
4	5	4
5	7	2
6	7	4



AGV Information	Origin -	Path -	Destination (Time)
AGV #1	1 (0)	2 (3)	4 (6)
AGV #2	3 (0)	5 (4)	7 (6)
New AGV	5		2

Figure 1: Example Problem.

It is of some consolation to note that since the entire problem can be modelled as a complex integer math programming problem, it is classified as an NP-hard problem (see Worst Case Behavior in Chapter 6). NP-hard, or nondeterministic-polynomial-time problems are a large class of problems having an important characteristic, namely that all algorithms currently known for finding optimal solutions to these problems require a number of computational steps that grows exponentially with the size of the problem. The NP-completeness discovery changed the direction of research on many difficult problems. Earlier research efforts were toward optimal solutions, but more recent efforts have turned to the heuristic algorithms, a more fruitful direction of determining near optimal solutions in polynomial time. Therefore, these heuristic methods become the logical solution route for this problem. As will be presented in Chapter 2, no methods currently exist to provide optimal solutions to the multiple AGV bi-directional routing problem, so an alternative method will be developed.

Heuristic algorithms utilize comparatively simple methods to obtain acceptable solutions to problems which are most often unsolvable or impractical to solve using exact techniques. Application of a rather standard heuristic, the "greedy" heuristic, resulted in no feasible solution being found. The "greedy" heuristic finds the best route and assigns that route to the solution. In the example problem, Figure 1 on page 12, that particular route is node 5 to node 4 to node 2, and is infeasible because a collision occurs on arc 2-4.

A dynamic programming solution approach to calculate an optimal routing would use an optimal *n-1* vehicle routing as the basis for obtaining an optimal *n* vehicle routing. When only one vehicle is present, the one vehicle routing problem is certainly tractable; in fact, it is equivalent to the shortest path problem. While this method had some intuitive appeal, the result was far from satisfactory. The problem was the initial difficult problem had subproblems that were just as difficult.

The impracticality with the dynamic programming approach is that the route of any one of the previously scheduled *n* vehicles can be modified. Therefore the dynamic programming solution methodology is really finding an optimal *n* vehicle routing at each step, rather than simply finding the best *one more* vehicle routing, as was hoped. A modification of any of the existing *n-1* routes has to be prevented to make the problem solvable.

Therefore the solution method used in this work requires one important assumption,

> The *n* AGV route will be calculated from an existing *n-1* AGV route by computing the optimal route for an additional AGV without modifying any of the existing AGV's routes.

Once this assumption was added, the dynamic programming approach becomes a feasible solution procedure. With this assumption, an effective heuristic method for calculating an *n* AGV route could be obtained by adding one AGV iteratively *n* times. This solution methodoiogy would also be very effective for real-time operations where AGV requests are received and routing directions are required. Real-time operation amounts to adding one more AGV to an existing network. Using the dynamic programming approach one vehicle would be added, along with its corresponding route, to an existing *n-1* vehicle routing, thus creating a new *n* vehicle routing.

The methodology presented in this work assumes no changes will be made to any of the existing *n-1* vehicle routes. While this might seem restrictive, it is not unrealistic. In any manufacturing operation, once a route is established, the vehicles and the goods they transport are expected to arrive at scheduled times. Any modification of an existing route could delay this schedule and create additional delays further down the production line.

An additional benefit of this methodology is the ability to develop "good" schedules for *n* vehicles. The application of the procedure iteratively *n* times would result in a "good" *n* vehicle schedule. The only additional requirement is the designation of an input order for the AGVs, but this order could represent the priority attached to each AGV in the material handling system.

Solution Methodology

Motivated by the challenge of directing multiple AGVs over a complex bi-directional network and the widespread applicability of this AGV control methodology to important "real-world" situations, the problem of developing a conflict resolution procedure for AGVs on bi-directional networks will be addressed. The objective of this study is to develop a procedure which will generate a locally optimal route for a new AGV through a bi-directional network without conflicting with any existing AGV traffic.

The remaining chapters of this work provide the background and details of this developmental effort. Chapter 2 contains a review of the literature on other potentially applicable methods and why these methods are unsuitable to this particular task. Chapter 3 provides a summary of the branch-and-bound solution methodology and a discussion of the exact conflict resolution procedure developed in this work. Chapter 4 presents proofs that the solution generated by the procedure is both feasible and optimal.

A cornerstone of the conflict resolution procedure is the shortest path algorithm. Chapter 5 provides a summary of shortest path algorithms and discusses the shortest path algorithm used in the example algorithm tested. Chapter 6 presents an example algorithm developed to test the conflict resolution procedure and the test problems generated to evaluate the algorithm. A discussion of the worst case and average performance of the test algorithm is also presented. Finally, some concluding remarks and recommendations for future research in the study of future AGV systems are presented in Chapter 7.

Chapter 2

REVIEW OF APPLICABLE LITERATURE

Vehicle Routing and Scheduling Algorithms

Solomon [75] presented a nice summary of the vehicle routing and scheduling problem. The entire summer issue of Networks (Vol 11, No 2, 1981) was dedicated to the vehicle routing and scheduling problem. Garey, Graham, and Johnson [33] discussed the difficulty of scheduling algorithms in general, pointing out that most fall in the class of NP-Hard problems. Lenstra and Rinnoy Kan [58] showed the vehicle routing problem is NP-Hard and Solomon [76] showed the vehicle routing and scheduling problem with time window constraints is NP-hard.

The vehicle routing and scheduling problem can generally be stated as follows:

> Given a set of transportation tasks to be performed, each with an origin, a destination, an earliest start time and a latest start time; given also a transportation network of clearly defined structure, and a set of vehicles of given speed; find an optimal sequence of tasks, their start times, and routes, such that the sum of the cost of vehicles required and the total cost of traveling are minimised. [16]

There are three main groups associated with this class of problem:

1. Vehicle routing problems.
2. Vehicle scheduling problems.
3. Dial-a-ride problems.

Vehicle Routing Problems

The vehicle routing problem is a very important problem in distribution management. Vehicles are routed over a transportation network with demands for services at various points. The decision concerning which path the vehicle travels while satisfying these demands is classified as a routing problem and can be described as follows:

Given a set of demand points and a central supply depot (or depots), find a minimum cost set of tours which meet the required demands subject to constraints on the capacity and range of the vehicles. A number of complicating constraints may also be present, such as special vehicle characteristics and crew scheduling constraints. [8]

The following formulation has been referred to as the generic vehicle routing problem.

$$\text{Minimize} \sum_{i=1}^{n} \sum_{j=1}^{n} \sum_{v=1}^{NV} c_{ij} x_{ij}^{v}$$

subject to

$$\sum_{i=1}^{n} \sum_{v=1}^{NV} x_{ij}^{v} = 1 \qquad j = 2, \ldots, n \qquad (1)$$

$$\sum_{j=1}^{n} \sum_{v=1}^{NV} x_{ij}^{v} = 1 \qquad i = 2, \ldots, n \qquad (2)$$

$$\sum_{i=1}^{n} x_{ip}^{v} - \sum_{j=1}^{n} x_{pj}^{v} = 0 \qquad v = 1, \ldots, NV; \; p = 1, \ldots, n \qquad (3)$$

$$\sum_{i=1}^{n} d_i \Big(\sum_{j=1}^{n} x_{ij}^{v} \Big) \leq K_v \qquad v = 1, \ldots, NV \qquad (4)$$

$$\sum_{i=1}^{n} t_i^{v} \sum_{j=1}^{n} x_{ij}^{v} + \sum_{i=1}^{n} \sum_{j=1}^{n} t_{ij}^{v} x_{ij}^{v} \leq T_v \qquad v = 1, \ldots, NV \qquad (5)$$

$$\sum_{j=2}^{n} x_{ij}^{v} \leq 1 \qquad v = 1, \ldots, NV \qquad (6)$$

$$\sum_{i=2}^{n} x_{ij}^{v} \leq 1 \qquad v = 1, \ldots, NV \qquad (7)$$

$$X \in S \quad \text{and} \quad x_{ij}^{v} = 0 \text{ or } 1 \qquad \text{for all } i, j, v.$$

where n = number of nodes ; NV = number of vehicles ; K_v = capacity of vehicle v ; T_v = maximum time allowed for a route of vehicle ; d_i = demand at node i ($d_1 = 0$) ; t_i^{v} = time required for vehicle v to deliver or collect at node i ($t_1^{v} = 0$) ; t_{ij}^{v} = travel time for vehicle v from node i to node j ($t_{ii}^{v} = \infty$) ; c_{ij} = cost of travel from node i to node j ; $x_{ij}^{v} = 1$ if arc i - j is traversed by vehicle v, 0 otherwise ; X = matrix with components $x_{ij} \equiv \sum_{v=1}^{NV} x_{ij}^{v}$, specifying connections regardless of vehicle type ; and S represents the set of subtour breaking constraints.

The objective function is to minimize total cost. Constraints (1) and (2) ensure that each demand node is served by exactly one vehicle. Route continuity is represented by equations (3), i.e. if a vehicle enters a demand node, it must exit from that node. Equations (4) represent vehicle capacity constraints and equations (5) are the total elapsed route time constraints. Equations (6) and (7) guarantee that vehicle availability is not exceeded.

Solution Strategies

Most solution strategies for vehicle routing problems can be classified into one of the following approaches:

1. cluster first - route second
2. route first - cluster second
3. savings / insertion
4. improvement / exchange
5. mathematical programming based
6. interactive optimization
7. exact procedures.

e following section contains a short summary of each of the above listed approaches:

Cluster first - route second

Cluster first - route second procedures group or cluster demand nodes or arcs first and then design economical routes over each cluster as a second step. Effective implementation of this idea was provided by Gillett and Miller [36] and Gillett and Johnson [35].

Route first - cluster second

Route first - cluster second procedures work in the reverse order. First, a large usually infeasible route or cycle is constructed that includes all of the demand nodes or arcs. Next, this large route is partitioned into a number of smaller, feasible routes. This approach has been used for effectively routing school buses and street sweepers [10].

Savings or Insertion

Savings or insertion procedures build a solution in such a way that at each step of the procedure (up to and including the next to the last step) a current configuration, that is possibly infeasible, is compared with an alternative configuration that may also be infeasible. The alternative configuration is one that yields the largest savings in terms of some criterion or that inserts, least expensively, a demand entity not in the current configuration with the existing route or routes.

Examples of savings/insertion procedures for single depot node and arc routing problems are described by Clarke and Wright [12] and Golden and Wong [40].

Improvement or exchange

Improvement or exchange procedures always maintain feasibility and strive towards optimality. At each step, one feasible solution is altered to yield another feasible solution with reduced cost. This continues until no additional cost reductions are possible. Examples of this methodology include the well-known branch exchange heuristic developed by Lin and Kernighan [60]. Their procedure started with any random feasible solution, sequentially examined *k* pairs of links for exchange such that the solution was improved and remained feasible. An obvious advantage of this procedure is the algorithm can be terminated at any time with a feasible solution available.

Mathematical programming approaches

Mathematical programming approaches include algorithms that are directly based on a mathematical programming formulation of the underlying routing problem. An example of the mathematical programming based procedure is given by Fisher and Jaikumar [26]. They formulated the vehicle routing problem as a mathematical program in which two interrelated components are identified. One component is a traveling salesman (routing) problem and the other is a generalized assignment (packing) problem. Their heuristic attempts to take advantage of the fact that these two problems have been studied extensively and powerful mathematical programming approaches for their solutions have already been devised. Other mathematical programming based methods include dynamic programming approaches for obtaining lower bounds in a variety of combinatorial optimization problems.

Interactive optimization

Interactive optimization is a general-purpose approach in which a high degree of human interaction is incorporated into the problem-solving process. The idea is that the experienced decision-maker should have the capability of setting and revising the optimization model based on subjective assessments and intuition. This almost always increases the likelihood that the model will eventually be implemented and used.

Exact Procedures

Exact procedures for solving vehicle routing problems include specialized branch-and-bound and cutting plane algorithms. The conflict resolution algorithm developed in this study would be classified as an exact procedure to solve a restricted version of the *n* vehicle routing problem.

Shortcomings

When trying to deal with the routing of AGVs over bi-directional flow networks the first four methods mentioned do not consider the possibility of opposing traffic conflicts while allowing an AGV to follow in close proximity behind another AGV. The mathematical-programming-based approaches generally use traveling salesman problems or generalized assignment problems as the basic formulation. Traveling salesman problem formulations are overly restrictive on the vehicle's route. These solutions force the visitation of each node rather than finding the shortest time routing. Generalized assignment problems simply assign routes to meet demand requirements. The problem with the assignment procedure is which routes should be considered for assignment. Both formulations also suffer from the inability to allow duplicate use of an arc when both AGVs are traveling in the same direction while prohibitting the duplicate use of an arc by AGVs traveling in the opposite direction.

The interactive optimization approach requires one of the other five methods to provide solutions for the decision maker to evaluate. Since each of the other five methods has shortcomings the interactive procedure suffers from similar shortcomings.

Vehicle Scheduling Problems

Vehicle scheduling problems can be thought of as a routing problem in which explicit consideration is given to the times at which various locations are visited. The vehicle scheduling problem is used to model transportation systems where delivery time constraints and precedence constraints are dominant. The problem can be stated as follows:

> Given a set of transportation tasks to be performed, each with an origin g_i, destination h_i, earliest start time b_i, latest finishing time e_i, and duration d_i a deadheading time t_{ij} is defined as the travel time from the end of task i to the beginning of task j. Find a schedule which minimises the number of vehicles required and/or the total travel time. Since the task durations are fixed, the minimisation of travel time is equivalent to the minimisation of deadheading time. [15]

A hierarchy of scheduling problems has been summarized in an excellent article by Bodin and Golden [8].

The vehicle scheduling problem was first described by Dantzig and Fulkerson [17] in 1954. They presented a linear programming formulation and used a simplex algorithm to solve it. Bellmore, Bennington, and Lubore [6] considered the scheduling and routing of a fixed fleet of nonhomogeneous vehicles to make a prespecified set of shipments each of whose delivery dates was restricted to be within some interval of time (time window). Their formulation had the ability to consider differences in speed, carrying capabilities, and operating costs of the vehicles. They used a network representation for this problem, where nodes represented departures or arrivals of shipments at particular times and arcs represent shipments. Using this representation, every departure or arrival must be represented by a different node for each possible departure or arrival time. The formulation has a very large number of variables for reasonably sized problems.

Levin [59] formulated models to minimize fleet size for either fixed or variable schedules using an integer program of the max-flow problem with "bundle-constraints." Tasks were represented by source-sink node pairs with all node pairs for a particular task "bundled" together to represent the time window for the task. Only one pair was chosen from any bundle. A branch-and-bound algorithm was used to solve a 26 trip problem in two minutes.

The vehicle scheduling problem with time windows was analyzed by Orloff [65]. He formulated the problem as a network problem where tasks were defined as nodes and deadheading time was defined on feasible arcs. Cost factors for travel time and waiting time and fixed transportation costs were included. His formulation was equivalent to an assignment problem and his solution used the following heuristic algorithm. Matching problems were solved to build up schedules by optimally matching single tasks with each other to form pairs and singletons. These schedules were combined to form larger schedules, and so on, until no more optimal matchings were possible. Improvements were then attempted by 3-optimum exchanges. His heuristic worked rapidly and effectively for some school bus scheduling applications. The vehicle scheduling methodology has some similarities to the AGV routing problem. The same problem variables are present, but there is a major shortcoming.

Shortcomings

Almost all the vehicle scheduling problems can be formulated as optimization problems on appropriately defined networks. In the case of the single depot vehicle scheduling problem, for example, this formulation leads to an efficient solution using a minimum cost flow algorithm. When applied to the AGV routing problem, the shortcoming of this solution methodology is at the heart of the minimum cost flow problem. Any flow constraint in the network must limit the flow along any arc to one unit. This constraint prohibits one AGV from following another AGV over the same arc, which may be the optimal feasible solution. A constraint is needed which will allow following traffic while at the same time prohibit opposing traffic. Without this constraint or group of constraints an optimal solution could be eliminated from feasibility or an infeasible solution selected as optimal.

This is not a simple process. One method is to create additional variables, one for each time segment along each track and then create additional constraints for each uni-directional and bi-directional arc. This really does not solve the problem, it only creates another problem, the already large number of constraints and variables grows enormously, and no method is available to solve the resulting problem.

Dial-a-Ride Problems

The dial-a-ride problem is concerned with the dynamic routing of vehicles and can be stated as follows:

> Given a set of origin-destination pairs of locations, determine a route which visits all the locations, in the correct order such that travel distance and customer waiting time are minimised. [16]

The dial-a-ride problem is an approximate model for a transportation system in which a request for service is made and an origin-destination pair is specified. Some models include limit times on pick-up or delivery, but in all models the objective is to provide the service as soon as possible.

The school bus scheduling problem is about the most data intensive routing and scheduling problem encountered by Bodin [9]. The routing component is a variant of the single depot routing problem and is usually solved using the route first - cluster second approach. The scheduling component organizes the partial routes into schedules. The starting and ending times for each of the routes must be specified along with the maximum allowable travel time. The objective used in the scheduling component is to minimize the total travel time of the vehicles.

While this formulation might seem similar in concept to the n AGV routing problem, the same shortcomings were present in this methodology as are present with the vehicle scheduling problem. The inability of this formulation to handle the detection of a conflict while allowing an AGV to follow another AGV is a problem all these algorithms cannot easily handle.

Shortcomings

The formulation listed in this section, in general, fail due to one of the following four circumstances:

1. The algorithms fail to provide collision detection provisions and it is not a trivial task to insert them.
2. Algorithms that provide time window constraints, i.e. collision avoidance, force the vehicles to be dispatched from a central point, a depot or warehouse.
3. Algorithms with the ability to handle the above two problems will not allow a vehicle to follow another vehicle along a path.
4. Algorithms which might be modified to address the above three problems require variables indexed to travel time. This indexing causes the already large number of variables to increase beyond the capability of currently available computer systems.

Airplane and Aircrew Scheduling Problems

The scheduling of aircrews has a structure which is very similar to that of the vehicle scheduling problem, but this problem is more complex because of the workrules and costs which govern the formation of crew schedules. Since the operating costs for the planes are several times greater than the crew costs, the scheduling of the planes is carried out first. The solution methodology is generally broken down into two actions - generating pairings and constructing bid lines. A pairing is a collection of trips that a crew must complete, with the trips beginning and ending at the same location. The bid lines are sets of pairings that represent the monthly work schedules for the crews.

When the complication of a time window in which a task must be carried out is introduced, this problem resemble the *n* AGV routing problem. But as with previous formulations, the ability of an AGV to follow another on an arc but not travel in opposing directions along an arc, makes this particular methodology unacceptable.

Multicommodity Network Flows

Using surveys by Kennington [49] and Assad [3], the multicommodity network flow solution methodology was checked for application to this problem. These multicommodity network flow problems arise naturally in network modeling wherever commodities are to be transferred from certain nodes to some other nodes in the network. These commodities could represent vehicles in a transportation system.

While some formulations closely parallel the basic requirements of the *n* AGV routing problem, a common shortcoming appears again. All the standard formulations define bi-directional arc flow variables between node i and node j as x_{ij} and x_{ji}. The formulation has a constraint $x_{ij} + x_{ji} \leq 1$ to prevent both directions of this bi-directional arc from becoming active anytime during network operation. One problem with this formulation is this constraint prevents a vehicle from following an existing AGV on an arc, since the constraint places an implicit constraint on each variable, $x_{ij} \leq 1$ and $x_{ji} \leq 1$. Another problem is this constraint should be time dependent. For example, if a bi-directional arc exists between nodes 1 and 2 and has length 3 and at time 0 half of that arc, the directed arc (2,1) becomes active. In this formulation the opposite direction arc (1,2) should only be blocked until time 3, but the arc (1,2) should remain unblocked for traffic following except for a one time unit reservation to prohibit simultaneous (piggy-back) travel.

The multicommodity flow network formulations examined to date do not resolve these problems without discretizing the time interval and indexing the variables over time. Additional constraints would be required to handle the different constraints. However, this indexing creates an tremendous growth in the number of variables, even for small problems. When the example problem, Figure 1 on page 12, is put in shortest path formulation there are 18 *0-1* variables and 7 constraints. Put in multicommodity network formulation the variable count exceeded 5000 and the constraint count exceeded 1500. While this number is manageable on most computer systems, it must be pointed out this example problem is very small and a limited time horizon was considered. As soon as a problem with more nodes and arcs is attempted and a longer traveling time horizon needs to be segmented, the number of variables really explodes.

Math Programming

The field of math programming is extremely broad. Many of the previously described problems in this chapter could be classified as math programming formulations. One unique math programming model discovered was a railroad routing problem designed to handle the meet-pass situation. This problem has a high degree of similarity with the AGV problem considered in this work. The meet-pass formulation is a mixed integer math program with *0-1* variables representing arc choices. This model formulation can be explained using Figure 2 [25].

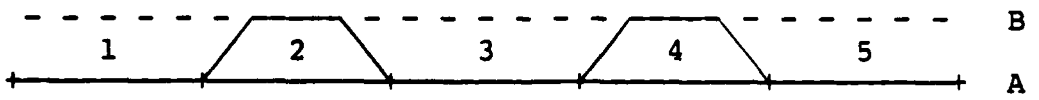


Figure 2: Meet - Pass Example

Note that the track has been divided into five separate regions corresponding to decision points along the track. For each active train there is a continuous decision variable, d_{Ajk}, which denotes the time which train k spends on track A in region j. Similarly d_{Bjk} denotes the time which train k spends on track B in region j. Constants l_{Ajk} and l_{Bjk} denote lower bounds on these times given that train k travels on track A or B in region j. To complete this example, use two trains traveling in opposite directions beginning at times t_1 and t_2 respectively. Let t_{jk} represent the time train k arrives at region j (this simplifies the presentation but is not necessary in the actual model). The transformed variables in this small example formulation are listed below:

$$t_{21} = t_1 + d_{A11} \qquad t_{42} = t_2 + d_{A52}$$
$$t_{41} = t_1 + d_{A11} + d_{B21} + d_{A31} \qquad t_{22} = t_2 + d_{A52} + d_{B42} + d_{A32}$$

This problem will generate a meet at either region 2 or region 4. Using a binary variable (m) to handle the either or constraint along with a large number (L). The final formulation is:

$$\begin{aligned}
t_{21} - t_{22} &\le L(1\text{-}m)\\
t_{22} + d_{A22} - t_{21} - d_{B21} &\le L(1\text{-}m)\\
t_2 - d_{A22} &\le L(1\text{-}m)\\
t_1 - d_{B21} &\le L(1\text{-}m)\\
t_{42} - t_{41} &\le Lm\\
t_{41} + d_{A41} - t_{42} - d_{B42} &\le Lm\\
t_1 - d_{A41} &\le Lm\\
t_2 - d_{B42} &\le Lm
\end{aligned}$$

Extra $d_{Ajk} \ge l_{Ajk}$ constraints can be handled implicitly as upper bounded variables. Using an appropriate objective function such as the minimization of total travel time, the formulation is now complete. This approach may have promise for smaller examples but as the number of AGV's increases, the possible number of meets and passes increases and hence the number of variables increases "enormously." Another problem is as the number of possible meets and passes increase, the difficult constraints increase.

Time Constrained Traveling Salesman Problem

The time constrained traveling salesman problem (TCTSP) is a special case of the traveling salesman problem where the visit to each location must be made within specified time windows. The incorporation of time window constraints within the traveling salesman model may also be found in recent work on dial-a-ride problems. Work done by Baker [4] proposed a special model with no zero-one variables but the use of absolute value constraints. There is similarity between this model and the railroad meet-pass problem listed earlier.

$$\text{Minimize } t_{n+1} - t_1$$

subject to

$$\begin{array}{rll}
t_i - t_1 \geq d_{1i} & i = 2, 3, \ldots, n & \\
|t_i - t_j| \geq d_{ij} & i = 3, 4, \ldots, n; & 2 \leq j < i \\
t_{n+1} - t_i \geq d_{i1} & i = 2, 3, \ldots, n & \\
t_i \geq 0 & i = 1, 2, \ldots, n+1 & \\
l_i \leq t_i \leq u_i & i = 2, 3, \ldots, n &
\end{array}$$

Solutions may be obtained through the use of branch-and-bound procedures which are necessary to handle the absolute value constraints.

This procedure is overly restrictive to the *n* AGV routing problem as it requires the new vehicle to visit every node rather than simply completing a required route. Some of the ideas from this formulation and the math programming meet-pass example are incorporated in the solution procedure presented in this work.

Simulation

Simulation modeling is a technique of imitating a physical system witl computer programs. The major benefit of simulation is that it is easier to manipulate mathematical equations than it would be to manipulate the physical hardware to solve performance problems, especially when the physical system is installed and operating. Simulation has often been used to compare the performance of different control strategies, but very seldom is it effective in determining the best solution from all available solutions. Normally in simulation there are numerous options available, yet only a few are considered best and hence, only those few are simulated. In this problem there are too many paths to consider, making it too time consuming to run all the options.

The real problem with applying this methodology to the AGV problem is how to obtain the paths to simulate. One method might be to generate the k-shortest paths between the desired origin and destination and simulate these results for feasibility. This method, with $k = 1$, was employed by Spinelli [77] but produced inferior results to the methods presented in this work. He did not attempt simulations with values of k other that one, however. An additional problem with simulation is the problem of what to do when abnormally long delays occur and how to determine when and which alternate paths should be explored.

No matter what method is used to obtain the paths, the simulation program must have the capability to force the AGV to delay in order to resolve a conflict, if and when a conflict is discovered. This can be accomplished using the zone-blocking control strategy presented earlier and executed in simulation using the seize-release resource techniques. While the study by Spinelli [77] addressed how ineffective simulation was in obtaining the best *n* AGV routing, it pointed out another interesting fact. He found zone segmentation had significant impact on AGVS performance on both uni-directional and bi-directional systems. He used a simulation model to analyze the effect of segment length on the overall performance of an AGVS.

An AGV Routing Algorithm

In 1987 Fujii and Sandoh [32] considered the control of many AGVs over a complicated network in a flexible manufacturing system with the object of providing minimum interferences and idle times. Their routing algorithm attacked the problem by first solving shortest path problems in the AGV network and then creating an assignment problem which assigned vehicles to these paths. They attempted to deal with the case where all *n* vehicles received requests for routing when they were on nodes and the vehicles could all start traveling simultaneously. It is interesting to note that, to this author's knowledge, this was the first published attempt at controlling multiple AGVs over bi-directional networks using an analytic method and not simulation methods. Their algorithm follows:

Step 1: For each vehicle, find the k^{th} shortest fundamental path from the current node to the requested destination node for $k=1,...,K$, where the fundamental path signifies a path that does not consider stops to avoid mutual interferences but simply consists of a sequence of nodes. Let $g=1$ and let $T=\infty$.

Step 2: Focusing on the traveling time of n vehicles, find a combination of n fundamental paths, one path for each vehicle, which presents the g^{th} shortest total traveling time among all combinations possible. Call this combination the g^{th} shortest fundamental path set. In the following, $R_p^{(g)}$ denotes the fundamental path for vehicle p $(p=1,...,n)$ in the g^{th} shortest fundamental path set.

Step 3: Compare T with t_g, where t_g is the total traveling time for $R_p^{(g)}$, $p=1,...,n$ and if $t_g \geq T$, go to Step 6, otherwise to Step 4.

Step 4: Assign stops to $R_p^{(g)}$, to avoid mutual interferences considering the unfinished part of the path from the previous request so that the total traveling time for n vehicles becomes the minimum. Create a linear programming formulation of an assignment problem to match stops to vehicles with the objective of minimizing the total stopping time. Let $\hat{R}_p^{(g)}$, denote the result for vehicle p.

Step 5: Compare T with $\hat{t}_g$ where $\hat{t}_g$ is the traveling time for $\hat{R}_p{}^{(g)}$, $p=1,\ldots,n$. If $\hat{t}_g < T$, then let $T=\hat{t}_g$ and $R_p^{*(g)} = \hat{R}_p{}^{(g)}$ for $p=1,\ldots,n$ and go to step 2 with $g=g+1$. Otherwise, go to Step 6.

Step 6: Stop. The optimal path set consists of $R_p^{*(g)}$ for $p=1,\ldots,n$.

While this algorithm has some very interesting characteristics and a certain similarity with the work presented in this study, there are two major differences. First, this algorithm deals with the entire network not simply adding one more vehicle. The problem they dealt with was the problem initially proposed which was determined to be much too difficult. Second, this algorithm is computationally expensive and limited in its application. Their concluding remarks seem to justify the conclusion made earlier in this work, the multiple AGV routing problem is very difficult and not computationally tractable. For example they evaluated a 12 node 12 arc network and had to solve 57,984,682,496 linear programs (LP) with an average of 32.17 constraints per LP. And this is only tracking one AGV. This is unbelievably complex and hardly able to provide real time control of an AGV network. Fuji and Sandoh offered the following concluding remarks, (the italic comments are provided by this author).

1. The number of linear programming problems is originally enormous, but may be reduced by introducing the idea of the interference candidate path and by eliminating the infeasible and inefficient problems. *This is a nontrivial task and may take more time than it saves. For the cited example problem* (12 node and 12 arc) *469 LP problems were eliminated as infeasible and 7 LP problems were eliminated as inefficient. This represents less than a .00000821 percent reduction.*

2. The computational time is long when many vehicles are installed in a simple and small-scale network but can be reduced by enlarging the size of the network while keeping the number of vehicles constant. This indicates the necessity for considering the adequate number of nodes and arcs in the network with respect to the number of vehicles installed. *As more AGVs are added the number of LP problems increases ; however, the assignment problems become simpler. Their conclusion was verified by* [77]. *He demonstrated that the more nodes in a network, the more opportunities to pass existed and the more close to optimal the resulting routing.*

3. The time to reach the optimal solution is much longer than the time required to obtain an initial feasible solution. Confirmation of optimality accounts for most of the computational time in this procedure. *This is true of most combinatorial*

algorithms. Since so much time goes into verifying optimality many algorithms maintain both upper and lower bounds on the optimal solution. When the gap between these two bounds is "small enough" the algorithm is terminated. The theory behind this idea is the extra work required to close the gap is not worth the effort.

An attempt was made to apply this methodology to the problem of adding a single vehicle to an existing AGVS. The result was very similar to the result the authors discovered and that are presented above. Consequently, this study presents the most promising technological advancement for the control of AGVs over bi-directional networks available.

Chapter 3

THE CONFLICT RESOLUTION PROCEDURE

Branch-and-Bound

Algorithms based on the branch-and-bound principle have proven useful for various *combinatorial optimization problems*. In essence, branch-and-bound methods are enumerative schemes for solving optimization problems. The underlying idea of a branch-and-bound algorithm is the decomposition of a given problem into several subproblems of smaller sizes and the evaluation of these, supposedly easier, subproblems. This decomposition principle is repeatedly applied to these subproblems until each undecomposed problem is either solved or proven not to yield an optimal solution to the original problem. The utility of the method derives from the fact that, in general, only a small fraction of the entire set of possible solutions are enumerated. All other remaining solutions are eliminated through the application of bounds which establish that these solution cannot be optimal.

The name "branch-and-bound" arises from its two main operations:

- *Branching* - which consists of dividing collections of sets of solutions into subsets.
- Bounding - which consists of establishing bounds on the value of the objective function over the subsets of solutions.

The branch-and-bound method was initially developed by Land and Doig [55]. Their technique has proven useful for solving integer, mixed-integer, and zero-one integer problems. Their branch-and-bound technique is described in more detail in the discussion that follows.

Suppose that the objective of some problem is to be minimized. Assume that an upper bound on the optimal value of the objective function is available. The first step is to partition the set of all feasible solutions into several subsets. Then, obtain a lower bound on the objective function for the solutions within each subset. Those subsets whose lower bounds exceed the current upper bound on the objective function value are then excluded (fathomed) from further consideration. Select one of the remaining subsets and further partition it into more subsets. The lower bounds for the partitioned subsets are obtained and used to exclude some of these subsets from further consideration. From all the remaining subsets select another one for further partitioning. Whenever a feasible solution is obtained the upper bound is updated. This process is repeated until a feasible solution is found whose corresponding objective function value is no greater than the lower bound on any subset. Such a feasible solution must be optimal since none of the subsets can contain a better solution.

Geoffrion and Marsten [34] provided a general procedure for solving difficult combinatorial problem. They claim the general algorithmic framework is based upon three key notions: separation, relaxation, and fathoming.

Separation. The set of feasible solutions for any optimization problem can be separated or partitioned into subproblems if the following conditions hold.

1. Every feasible solution to the original problem is a feasible solution of exactly one of the partitioned subproblems.

2. A feasible solution to any partitioned subproblem is a feasible solution to the original problem.

The most popular way of separating integer programming problems is by means of contradictory constraints on a single integer variable.

Relaxation. Any constrained optimization problem can be relaxed by loosening its constraints. This results in a new problem, the *relaxation.* Relaxations have the following properties:

1. If the relaxation has no feasible solution, the original problem is infeasible.

2. The minimal value of the original problem is no less than the minimum value of the relaxation.

3. If an optimal solution of the relaxation is feasible in the original problem, the solution is an optimal solution.

The most popular type of relaxation for integer programming is the dropping of the integrality requirements on all variables.

Fathoming. If it can be ascertained by some means that a subproblem cannot contain a feasible solution better than the best solution yet found, the subproblem can be dismissed from further consideration or fathomed. There are three general types of fathoming.

1. *Infeasibility.* If the relaxed subproblem being evaluated has no feasible solution, then the subproblem has no feasible solution. The subproblem need not be partitioned since it cannot possibly contain an optimal solution.

2. *Dominance.* If the lower bound of the subproblem, the optimal solution of the relaxed subproblem, is no better than the best feasible solution found so far, then the subproblem need not be partitioned. The subproblem need not be partitioned since it cannot possibly contain an optimal solution.

3. *Feasibility.* If the relaxed subproblems solution is feasible to the subproblem then the subproblem need not be partitioned.

If these three notions are put together into a general branch-and-bound algorithm, the following results.

General Branch-and-Bound Algorithm

Step 0: *Initialization step.* Set $Z_U = \infty$. Begin with the entire set of solutions under consideration as the only "remaining subset." This set of solutions is usually obtained through some relaxation. Note, this set includes all infeasible solutions that cannot conveniently be eliminated. Before iterating through the steps below, apply just the bound step, fathoming step, and stopping rule to this only remaining subset.

Step 1: *Branch step.* Use some branch rule to select one of the remaining subsets (those neither fathomed nor partitioned) and partition it into two or more new subsets of solutions.

Step 2: *Bound step*. For each new subset, obtain a lower bound Z_L on the value of the objective function for the feasible solutions in the subset.

Step 3: *Fathoming step*. For each new subset, exclude it from further consideration if

Fathoming test 1: *Infeasibility*. The subset is found to contain no feasible solutions,

or

Fathoming test 2: *Dominance*. $Z_L \geq Z_U$,

or

Fathoming test 3: *Feasibility*. The best feasible solution in the subset has been identified ($Z_L =$ objective function value); if this occurs and $Z_L < Z_U$, then reset $Z_U = Z_L$, store this solution as the incumbent solution, and reapply Fathoming test 2 to all remaining subsets.

Step 4: *Stopping rule*. Stop when there are no remaining subsets; the current incumbent solution is optimal. If there is no incumbent solution, then the problem possesses no feasible solutions. Otherwise, go to step 1.

Conflict Resolution Procedure

This work computes a minimum time routing for a new AGV. The new AGV is added to existing AGV traffic in a bi-directional network and must avoid conflicts with any of the existing traffic. Motivated by Young's challenge that directing multiple AGVs over a network with bi-directional pathways is "unmanageable" [81] and by the universal applicability this methodology would have in industry, a branch-and-bound based conflict resolution procedure is developed.

Using the underlying idea of branch-and-bound, the initial routing problem is relaxed by removing all existing traffic from consideration and an initial routing obtained. If this relaxed solution is not feasible, the problem is decomposed into one or two subproblems. Each decomposition is evaluated until either a feasible solution is found or it is proven the decomposition cannot yield an optimal solution. This approach is incorporated in the following branch-and-bound procedure. The procedure will be presented and then followed by a discussion of each step in more detail.

Step 1: *Initialization.* Set $Z_U = \infty$. Solve the relaxation of the initial problem, i.e. find the shortest path from the origin to destination without regard to any existing traffic. If no path exists, then no feasible solution exists and the procedure ends, otherwise, evaluate the path for feasibility, i.e. see if this routing conflicts with the existing traffic. If this shortest path is feasible, it is optimal and the procedure ends. If not go to step 2b.

Step 2: *Branch and Travel.* Remove a solution from the active list and check it for feasibility. If the active list is empty go to step 3.

- a. If no conflict exists, a feasible solution has been found. Update Z_U and return to Step 2.
- b. If a conflict is found, generate two alternative solutions:
  - b1. Find the shortest path from the origin to the destination over a modified network with the conflicting arc removed. It is possible no shortest path exists in this modified network.
  - b2. Determine the minimum delay time necessary to resolve the conflict.

Place the solution(s) found on the active list and return to Step 1.

Step 3: *Termination.* If any feasible solution was found, Z_U represents the optimal travel time between origin and destination without causing a conflict to occur.

Initialization

The first step of the procedure is the initialization phase. All network information, the node, arc, and cost data, as well as the existing AGV routing information, and the new AGV's origin and destination is input. The first computational step is to obtain an initial shortest path solution from the new AGV's origin to its destination. Realize, however, this shortest path solution may not be a feasible routing because other existing AGV's routes may conflict over the use of some arc along the calculated shortest path. If this path is determined to be a feasible solution, it is an optimal solution. If no shortest path can be found then the original problem is infeasible. This initialization step is performed only once.

Branch and Travel

The second step of the procedure is the heart of this branch-and-bound methodology. This branch and travel step is repetitive and performed until the active list becomes empty. Each solution on the active list represents a lower bound calculated for a set of paths.

The optimal solution to the original problem would be obtained if all possible paths were evaluated and the shortest conflict free route was selected. However, previous discussions in Chapter 2 demonstrated the difficulty that task presented. Solving the relaxed subproblems or calculating the minimum route in each partitioned set of paths provides a lower bound for the optimal objective value. As additional constraints or conflicting arcs are identified, a repartitioning of the set containing the conflicting path results in restricted relaxations. Each lower bound solution of a partitioned set represents a relaxation of the original problem differing only in the restrictions used while obtaining that particular solution.

A lower bound solution for a partitioned set of paths is chosen for evaluation. This minimum route is checked for feasibility by simulating the movement of the new AGV along the entire route, checking for an opposing or simultaneous travel conflict. If no conflict is found the solution is feasible and fathoming test 3, *feasibility*, is applied. This feasible solution is compared with $Z_{U,}$ the best solution obtained so far. If this solution is better, Z_U is replaced by the feasible solution found. If not the entire set is removed from future consideration.

If a conflict is discovered then this particular set of paths will be repartitioned into three subsets. One subset represents all the paths in the set that do not cover the detected conflicting arc. Another subset represents all the paths in the set that travel over the detected conflicting arc outside the detected conflicting time window. The remaining subset contains all the paths traveling over the conflict arc within the conflicting time window. Clearly this partitioning divides the "parent" set into three mutually disjoint subsets whose union is the "parent."

Restating the previous paragraphs using mathematical set notation, let S represent the set of all paths for consideration in the original problem. S = { Set of all minimum delayed feasible paths from the new AGV's origin to its destination }, where minimum delayed feasible paths imply there exists no purposeless idle time along the path. To solve the problem formulated in this work search the set S for the minimum path. Let T = { Set of all paths from the new AGV's origin to its destination }, therefore T represents a relaxation and is generated without regard to any existing network traffic. The set T represents feasible solutions to the initial relaxation of the problem. Clearly S is a subset of T. Define Z, the objective function, as the minimum arrival time at the new AGV's destination. Then if the minimum objective value in T, occurring using path π^*, is also an element of S, the procedure is complete and no branching is required. However, if that solution, π^* is not feasible then T needs to be partitioned.

Let T^1 represent the set T referred to above and let T^{2i} = { set of all paths from the new AGV's origin to its destination that do not use the detected conflict arc } and T^{2i+1} = { set of all paths from origin to destination that use the conflict arc, but the new AGV's time interval on the detected conflict arc is outside of the time interval where the conflict arc was used by the conflicting existing AGV }. Let F = { set of all paths from origin to destination that use the conflict arc, but their time on the conflict arc is within the time interval identified }. Clearly the union of these three partitioned sets represents all of T^1. Using fathoming test 1, *infeasibility*, the set F can be fathomed. There cannot exist a feasible solution to the original problem in F since there will always be a conflict over the detected conflict arc. Therefore, the set T^1 can be replaced by two sets, T^{21} and T^{21+1}.

The lower bound of the set T^1 is evaluated for feasibility. When a conflict is found the set T^1 is replaced by T^{21} and T^{21+1}. Lower bounds of these two sets may not need to be added to the active list. Step b1 computes lower bounds for each of the two subsets and may result in either one or both of the subsets being fathomed. For example, it is possible that removing the conflicting arc disconnects the origin from the destination. In that case, subset T^{21} is the empty set and has no shortest path solution. Fathoming test 1, ***infeasibility***, allows the empty subset T^{21} to be fathomed. Subset T^{21+1} can not be the empty set so step b2 computes a lower bound for this set, ${Z_L}^{21+1}$. If this lower bound is greater than Z_U, the best solution discovered so far, ${Z_L}^{21+1} > Z_U$, then fathom test 1, ***dominance***, is applied. No further evaluation of this set is required.

The lower bound of each partitioned set is added to the the active list. The active list is rexamined, a lower bound of a partitioned set chosen, and the process repeated until the active list becomes empty. An empty list implies that all problems have been implicitly evaluated and either eliminated for infeasibility, suboptimality or feasibility. In the case of feasibility, the set's lower bound is compared with the best previously obtained solution and the least value retained. Once the active list becomes empty the evaluation process ceases.

Termination

The repetitive decomposition and evaluation phase has been concluded with all possible solutions implicitly enumerated. The incumbent solution, Z_U, the best solution found so far, represents an optimal solution. If no feasible solution was found then the problem was infeasible.

Chapter 4

PROOF OF FEASIBILITY AND CORRECTNESS

This study has developed a procedure for the control of automated guided vehicles in a complex AGVS. This control procedure develops a minimum time routing for adding a new AGV to an existing AGVS. Applications for this procedure seem widespread.

Wenzel [80] discussed modifying material handling systems when AGV's delay time increased due to routing problems, frequently reaching between five and ten minutes wait between vehicles. Gould [42] acknowledged that all major manufacturers have installed systems with the capability of operating the AGVs in either direction, yet he was unable to identify a manufacturer operating the AGVs in a bi-directional capacity. Schwind [71] pointed out the bi-directional capability is used for a short run in and out of a pickup or drop off station which is feeding a multi-aisle automated storage/retrieval system. While he pointed out Raymond Corporation uses the technology for some simple shuttle systems he claims few situations need bi-directional capability. Upon closer evaluation and a later conversation [72] he acknowledged the reason there were no major applications of this technology was there existed no control technology available for dealing with the inevitable traffic conflicts that would result. Yet Egbelu [24] firmly demonstrated improved material handling capabilities using the bi-directional technology, if routing control could be provided.

The procedure developed in this work provides such a control mechanism. To justify that claim the following sections present two important proofs. The first proof is that the path determined by the conflict resolution procedure is a feasible conflict free routing for the new AGV. The second proof is that the route is a minimum elapsed time routing for the new AGV from its origin to destination.

Proof of Feasibility

Theorem: *The route generated by the conflict resolution procedure is a feasible path,* i.e. there is no use of an arc by the new AGV that conflicts with the use of that same arc by an existing AGV.

There are two types of paths in an AGV network, uni-directional paths and bi-directional paths. Uni-directional paths cannot, by definition, have a collision occur from opposing traffic, however, simultaneous travel (piggy-back) can occur and must be prevented. If different speed AGVs were used overtaking traffic would become a problem but, in this work, all AGVs are assumed to be traveling at the same speed. Bi-directional path traffic needs to be checked for simultaneous travel as well as for opposing traffic conflicts.

Observation 1: *There are only two types of infeasible paths possible.*

An infeasible path occurs when one of the following conflict conditions is present:

1. Two AGVs attempt to simultaneously travel in the same direction along the same arc (piggy-back travel).

2. Two AGVs attempt to simultaneously use a bi-directional arc in opposite directions (opposing traffic).

Simultaneous use of an arc, provided adequate separation exists, in the same direction is authorized and represents one advancement of this methodology over previously used simulation directed zone-blocking control methods.

Observation 2: *The conflict resolution procedure's route has no piggy-back travel conflicts.*

The first and simplest possible conflict to examine is the exact duplicate arc usage conflict (piggy-back travel). This conflict requires an exact match of arc entry node, arc exit node, and time entering the arc. This is a fairly straight forward check to make. A simple comparison of arc endpoints and arc entry times among existing AGV traffic against the newly routed AGV will provide detection of a piggy-back conflict. If these three duplications are found then a conflict is indicated, otherwise search for an AGV attempting to travel in the opposite direction of the new AGV on the same arc.

Observation 3: *The conflict resolution procedure's route has no opposing traffic conflicts.*

An opposing traffic conflict is detected by checking the new AGV's arc entry node against all existing AGV's arc exit nodes and if a match is found then check for the same arc entry node of that AGV against the new AGV's arc exit node. If the new AGV's path includes the requirement to travel in the opposite direction on an arc used in another AGV's path that does not necessarily mean there will be a collision but it certainly identifies an arc where a possible conflict could exist. Consider the numerical example pictured in Figure 3 to clarify this idea.

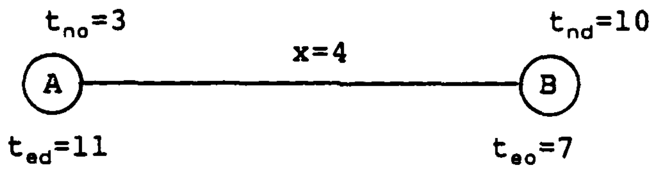


Figure 3: Numerical example of Conflict Arc Check

Is there a conflict ?

The new AGV, depicted traveling left to right along the top of the arc, departs node A at time $t_{no}=3$ and departs node B at time $t_{nd}=10$. An existing AGV, depicted moving right to left along the bottom of the arc, departs node B at time $t_{eo}=7$ and departs node A at time $t_{ed}=11$. To determine whether or not a conflict actually exists the computation of the arrival time of each AGV at their destination is required. Locating the conflict arc and its travel time provides the information necessary to compute destination arrival times.

In this case the travel time is 4, hence the new AGV arrives at node B at time 7 and the existing AGV arrives at node A at time $t_{na}=7$. In this particular case, see Figure 4, there is no conflict because the existing AGV departs node B at the same time the new AGV arrives. Since $t_{na}= t_{eo}=7$, a pass occurs at node B with no collision detected.

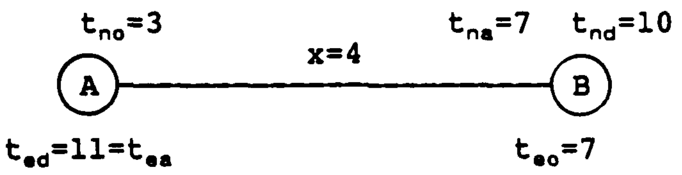


Figure 4: Numerical example of Conflict Arc Check with No Conflict

In terms of the efficient implementation of this opposite traffic conflict check, simply comparing arc exit node departure times is sufficient as the departure time very often equals the arrival time at that node. In cases where delays are present, for example after iterative applications of this methodology, the differences in arrival times are resolved by finding the travel time for the conflict arc and calculating the actual arrival time. The computational efficiency of this method for conflict checking is enhanced since the travel time over an arc only has to be found if a possible collision is discovered. Only in the case of a possible collision, will the search for the arc be accomplished along with the calculation of the arrival times, otherwise no arc search is necessary.

An actual conflict is discovered only if the new AGV's arc entry time is less than the existing AGV's arc exit time, i.e. the new AGV departs its node before the existing AGV arrives, and the new AGV's arc exit time is greater than the existing AGV's arc entry time, i.e. the existing AGV left its node before the new AGV arrived. Using the symbols of Figure 4, if $t_{no} < t_{ea}$ and $t_{na} > t_{eo}$ then a conflict occurred and appropriate action needs to be taken, i.e. repartition that solution set.

This conflict checking method allows the AGVs to proceed along their path when no conflict has been detected over the route examined. This progression continues until the entire route is checked or a conflict is found. As long as simultaneous travel and possible collisions do not occur, the conflict checking method allows the AGVs to proceed along their routes. If the procedure recognizes a situation where an existing AGV and the new AGV both want to utilize the same arc in opposite directions, then the second stage of identifying conflicts occurs.

PROOF: Assume, for the sake of contradiction, the solution provided by the conflict resolution procedure is not feasible. Observation 1 indicated the existence of only two possible infeasible paths in the AGV network. This provides only two cases for consideration:

Case 1. Observation 2 demonstrated that piggy-back conflicts were not possible in the conflict resolution procedure's solution. Therefore, by observation 1, the infeasibility must be an opposing traffic conflict, but that contradicts observation 3.

Case 2. Observation 3 demonstrated that opposing traffic conflicts were not possible in the conflict resolution procedure's solution. Therefore, by observation 1, the infeasibility must be a piggy-back conflict, but that contradicts observation 2.

Since both cases resulted in contradictions, the solution must be feasible. □

Proof of Local Optimality

Theorem: *The route generated by the conflict resolution procedure is optimal*, i.e. there is no new AGV routing from origin to destination that provides an earlier arrival time at its destination.

In the following section two propositions are provided to prove the theorem.

Proposition 1: *If a feasible solution exists, an optimal solution exists in the union of all paths represented on the active list.*

This proposition is established using induction on the active set at successive branching nodes. The hypothesis holds trivially for the first branching node (the initial relaxation), as the initial set of paths contains all routes from the new AGV origin to its destination, hence must contain an optimal path if one exists. If no path exists from origin to destination then there is no optimal solution.

The hypothesis is assumed true for the nth branching node and all previous branching nodes. It will be shown that the branching set that follows this step maintains the existence of an optimal solution.

Using the set representation of Chapter 3 for path subsets and the upper and lower bound variables, the active list at step n can be represented as $AL_n = \{ Z_i, Z_j, ..., Z_k, Z_U \}$; where $Z_L{}^j$ represents a lower bound for a set of paths T^j and Z_U is an upper bound on the travel time for the best path, π^U, obtained so far. The path represented by π^U, with travel time, Z_U is not really maintained on the active list. It is stored separately, but for convenience in the proof it will be considered as an element of AL_n^T, $AL_n^T = \{T^i, T^j, ..., T^k, \pi^U \}$. By assumption the union of all the path sets represented in AL_n^T, $\cup\{AL_n^T\}$, contains an optimal solution. It must be shown that the union of the set of paths at step $n+1$, $\cup\{AL_{n+1}^T\}$, contains an optimal solution. Let π^c be an optimal path contained in $\cup\{AL_n^T\}$.

The description of the conflict resolution procedure demonstrated that all subsets remain in AL_n^T except the one subset selected for evaluation. Assume without loss of generality, that $Z_L{}^j$, representing the lower bound for the set of paths, T^j, is selected at branching step $n+1$ for evaluation. During the evaluation of path π^{L_j} two cases may exist.

Case 1. No conflicts are discovered in the path $\pi^{L}{}_{j}$. If no conflicts are discovered in that path, all other paths in the set T^{j} are fathomed due to *dominance*. The lower bound of the set, $Z_{L}{}^{j}$, is compared with Z_U, with Z_U assigned the lowest value. Let the time to travel over path π^c be Z_c. Therefore if $Z_c \neq Z_{L}{}^{j}$, i.e. $\pi^c < Z_{L}{}^{j}$, then π^c must still remain in $\cup\{AL^T_{n+1}\}$.

Case 2. A conflict is detected in path $\pi^{L}{}_{j}$. If a conflict is detected, the set, T^j, must be partitioned. As previously described, the partitioning process results in three subsets. One of the subsets, F, is immediately fathomed for *infeasibility*. Therefore, the set T^j is replaced by two sets, T^{2j} and T^{2j+1}. Each subset, with its lower bound, is placed on the active list. Since the partitioning process only subdivides the set and eliminated a set containing only infeasible paths, an optimal solution still remains in $\cup\{AL^T_{n+1}\}$. There were no feasible routes lost in this partitioning and π^c must still be an element of $\cup\{AL^T_{n+1}\}$.

The conflict resolution procedure also includes another fathoming rule and this rule is now considered. Fathoming a set due to *dominance* means $Z_L \geq Z_U$. When a set is fathomed for *dominance* there can not exist a better solution. Since Z_U is retained, it follows that $\cup^T\{AL_{n+1}\}$ contains and optimal solution if $\cup^T\{AL_n\}$ does.

Proposition 2: *If a feasible solution exists, when there are no more lower bound solutions on the active list the optimal solution is determined.*

This is a fairly straight forward proposition to justify. Since the conflict resolution procedure has only a finite set of paths to search it must, at some point, find a feasible solution, if one exists. At every stage of the procedure at least one path and often a set of paths is fathomed. With this forced elimination of paths and only a finite number of paths to evaluate, the procedure must run out of sets to evaluate. The routine terminates when the list of candidate problems becomes empty and the list must become empty after a finite number of evaluations. At procedure termination, if a feasible solution has been found, i.e. $Z_U < \infty$, that solution is an optimal solution to the original problem.

PROOF: Proposition 1 and 2 establish the theorem. Since an optimal solution is never eliminated, by proposition 1, and since the procedure must terminate, by proposition 2, the procedure will terminate with an optimal solution, if one exists. $\square$

Chapter 5
SHORTEST PATH ALGORITHMS

Shortest path problems are by far the most fundamental and also the most commonly encountered problems in the study of transportation and communication networks. This is particularly true if the shortest path problem class is generalized to include related problems, such as the longest path problem, the most reliable path problem, the largest capacity path problem, the all shortest path problem, and various routing problems [37]. These mathematical models seek to improve efficiency and service by increasing capacity, reducing travel time, minimizing congestion, reducing the cost of transportation service, improving vehicle routing, or reducing energy utilization. With all these applications it is not surprising that a large number of papers, reports, and dissertations have been published on the subject of shortest path algorithms. There have appeared a number of excellent surveys, review papers, and bibliographies, such as those by Dreyfus [21] and Deo and Pang [19].

A review of past work shows algorithmic research has focused on the shortest path from one node to all other nodes for two reasons [39]. First, many of the applications require the repeated finding of shortest paths from one to all other nodes in order to accommodate changing criterion function coefficients or different criterion functions. Second, solution methods for mathematical programming problems as well as for other shortest path models often rely on iterative determinations of shortest paths from one node to all other nodes. This necessity for finding the shortest path several hundred or thousand times within one application of a general algorithm, has motivated researchers to seek the most efficient solution method for large directed networks. Since repeated determination of shortest paths in the AGV network forms the core of the conflict resolution procedure, the search for the fastest shortest path procedure is an important part of this research effort as well.

While the literature contains many shortest path algorithms, there are really only a handful of general methods for solving shortest path problems. Each general algorithm has within it subalgorithms, that is, special subproblems that must be handled in order to execute the general algorithm. Historically new algorithms were developed because researchers devised ingenious ways of handling one or more of these subproblems in a mathematically efficient manner. The power of computer implementation technology has demonstrated significant reduction in solution times for most shortest path problems. Dial, Glover, Karney, and Klingman [20] reported that improved implementation technology caused solution times for shortest path problems to drop from one minute to slightly more than one second, using the same general shortest path algorithm, computer, and compiler.

Shortest Path Algorithm Terminology

A network G(N,A) consists of a finite set N of nodes and a finite set A of arcs, where each arc $a \in A$ may be denoted as an ordered pair (u,v), referring to the fact that the arc is conceived as beginning at a node $u \in N$ and terminating at a different node $v \in N$. Let $\ell(a)$ or $\ell(u,v)$ denote the length associated with network arc $a=(u,v)$. A network may be represented in a computer in several ways and the manner in which it is represented directly affects the performance of algorithms applied to the network.

The most popular way of storing a network is to use a linked list structure. In this method, all of the arcs that begin at the same node are stored together and each is represented by recording only its ending node and length. A pointer is kept for each node heading and indicates the block of computer memory locations for the arcs beginning at this node. The set of arcs emanating from node u is called the *forward star* of node u and denoted by FS(u); i.e., FS(u) = { (u,v)εA }. If the nodes are numbered sequentially from 1 to $|N|$ and the arcs are stored consecutively in memory such that the arcs in the forward star of node u appear immediately after the arcs in the forward star of node u-1, then the forward star form requires only $|N|+2|A|$ units of memory. Figure 5 illustrates the storage of a network in sorted forward star form. The number in the square attached to an arc of the network diagram is the arc length. Most of the newer and faster algorithms assume that the network is represented in this forward star form.

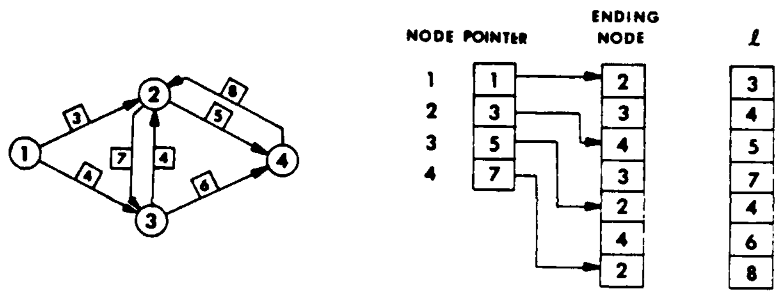


Figure 5: Forward Star Network Representation

Forward star forms for storing networks are commonly used with special algorithms called labeling methods for implementing shortest path and network flow solution procedures. In general, labeling methods are the most widely used methods for industrial and governmental applications because such methods are especially effective in application to large sparse networks [20].

A rooted tree, is a network $T(N_T,A_T)$ together with a node r, called the root node, such that each node of N_T is accessible from r by a unique arc-simple path in T. A common way of representing a tree in a computer is to think of the root node as the highest node in the tree and all the other nodes hanging below the root. The tree is then represented by keeping a pointer list which contains for each node $w \neq r$ in the tree, the starting node v of the single arc in the tree terminating at w. This upward pointer is called the predecessor of node w and denoted p(w).

Most labeling algorithms keep another list indexed by the node numbers and associated with the tree T. For each node v the list contains a label d(v), whose value is the length of the unique path from r to v in T. Nodes not in T may or may not be labeled with a node potential value, d(v). Usually the unlabeled nodes are given a very large value, usually represented as ∞, indicating that they are not yet reachable through the tree. The root r is assigned a node potential of zero. Labeling algorithms typically start with a tree, T, consisting only of the root node r and seek to enlarge and modify T until it becomes a shortest path tree of the original network G.

Labeling methods for computing shortest paths or minimum trees have been divided into two general classes, label setting and label correcting methods. The distinction is based on the order used for scanning nodes and whether the label of the scanned node can be declared as permanent or not. Both methods typically start with a tree $T(N_T,A_T)$ such that $N_T = \{r\}$ and $A_T = \emptyset$.

Label setting methods, as stated, start with an initialized tree, $T(N_T,A_T)$ with the distance label of the root set to zero and all other distance labels set to infinity. The nodes in the forward star of the root node, FS(r), are scanned (examined) to determine if any distance label can be improved. Each node whose distance label is improved is placed on an initially empty list called the *scan eligible*, (SE) node list. Along with the improved node the arc associated with the improved distance label is recorded. On each subsequent iteration the node with the minimum distance label on the SE node list is selected, removed, and added to N_T and its associated arc is added to A_T. This node is then scanned analogously to the root node and the SE node list is updated. A fundamental property of label setting algorithms is that T is always the shortest path tree for all nodes in N_T at each iteration. It follows that a label setting method will perform exactly $|N|$ iterations and will examine each arc exactly once. The algorithm terminates when SE is empty or after $|N|$ iterations.

Label correcting algorithms, as stated, start with $T(N_T,A_T)$. At each iteration, label correcting algorithms seek to improve T by reducing the distance label of at least one node in N_T by

1. adding a new arc and node to T
2. replacing an existing arc so as to change and shorten the path to a node already in T
3. updating a distance label without adding a new arc or node to T.

Label correcting algorithm do not choose scan nodes in a fixed order and their labels are not permanent until termination.

The primary difference between a label correcting algorithm and a label setting algorithm is the node selection criterion applied to the SE node list. Label setting algorithms select as the scanning node, the node from the SE list with the minimum distance label. Label correcting algorithms may select any node from the SE node list. Changing the node selection rule destroys the shortest path property of T and hence the algorithm may not have a polynomial bound. Empirical studies [20] indicate the effort required to find the minimum distance label often outweighed this disadvantage.

Comparison of Shortest Path Algorithms

In a study by Dial, Glover, Karney, and Klingman [20] five of the chief label correcting algorithms and four of the chief label setting algorithms were investigated. Each code was implemented and tested using efficient data structures and processing methods. For dense problems one label setting code proved best, while for sparser problems and problems of certain restricted structures, one label correcting code performed best. The margin of superiority in each case made it evident that no single code was close to dominating the others for all problems tested. The key conclusion was there was not a single code that could be applied to all circumstances and always provide the best result.

In an article by Glover, Klingman, and Phillips [38] the authors investigated a composite procedure which would make use of label setting ideas yet reduce the number of rescans like a label correcting procedure. They developed a polynomially bounded shortest path algorithm, called the partitioning shortest path (PSP) algorithm. Their new algorithm maintained the list of scan eligible nodes in two parts, a concept which is implicit in the label correcting algorithms. Nodes are added to the second part of the list, transferred to the first part when certain conditions are met, and then removed from the first part for scanning. To accomplish this process the algorithm calculates a number called a threshold value. Nodes in the second part of the SE node list are transferred to the first part when the distance labels are less than or equal to the threshold value. All nodes in the first part of the SE node list are then sequentially scanned.

The PSP algorithm explicitly identifies an efficient node scan procedure. Only the nodes whose distance labels have been improved are scanned and improved distance labels are used as soon as they are determined. By conceptually linking label correcting and label setting algorithms, this computationally efficient hybrid of the two approaches was proven to be polynomially bounded. Test results indicated the PSP algorithm dominated all of the leading label setting and label correcting codes.

The PSP Algorithm

The general steps of the Glover, Klingman, and Phillips [38] PSP algorithm are as follows:

Step 0: *Initialization.* Initialize the predecessor p(i), and distance labed d(i) for each node $i \in N$ as follows:

$p(i) = 0$ for all $i \in N$;
$d(i) = \infty$ for all $i \in N$, such that $i \neq r$;
$d(r) = 0$.

Set iteration $k = 0$.

Create two mutually exclusive and collectively exhaustive lists of scan-eligible (SE) nodes called NOW and NEXT. Initially set NOW = $\{r\}$ and NEXT = $\emptyset$.

Step 1: *Select an Element of NOW.* If NOW is empty, go to Step 3. Select any node u from NOW.

Step 2: *Scan Selected Node.* Scan node u by deleting it from NOW and examining each node v in the forward star of u, FS(u) (where FS(u) = $\{(u,v) \in A\}$) as follows: If $d(u) + \ell(u,v) < d(v)$, then redefine $d(v) = d(u) + \ell(u,v)$, $p(v) = u$, and add node v to the NEXT list if it is not already on NEXT or NOW. When all nodes in FS(u) have been examined, return to Step 1.

Step 3: *Repartition Scan Eligible Nodes.* If NEXT is empty, stop. Otherwise, set $k = k+1$ and transfer all nodes from NEXT to NOW and return to Step 1.

The key to the polynomial complexity of this algorithm is the partitioning of the SE node list. The fundamental property of partitioning is that it bounds the number of times that nodes are transferred from one part (NEXT) of the scan eligible list to the other part (NOW). This bound is possible because partitioning guarantees that at least one node has its distance label permanently set at each iteration under a variety of transfer criteria.

A distance label is called *sharp* if it represents the actual distance from r to i in the current tree and not just an upper bound. Glover, Klingman, Phillips, and Schneider [39] developed a variant of the PSP algorithm which attempted to maintain sharp labels naturally outside of the computationally expensive updating process and incorporated arc density information into the node scan rule. They created three partitioned sets instead of the previously described two. This key concept of the new algorithm restricts the node transfer from NEXT to NOW by providing an intermediate node set. While the authors claimed the three sets were mutually exclusive, their algorithm did not maintain this property.

The algorithm used in this work maintains near sharp labels and incorporates arc density information into the node scan selection rule. This algorithm also takes care to maintain the sets mutually exclusive property. By preventing overlapping evaluation of nodes, the algorithm's speed is enhanced.

Chapter 6

TEST ALGORITHM AND PERFORMANCE

An example algorithm was coded and tested. The algorithm that follows is a pseudo-code example of the tested conflict resolution algorithm used in this work.

```
        Call RDATA
        Call PSP
        Call FIPATH
        Call CONFCK(ans)
        If ans = no  then   STOP
          else Call PART
LOOP:   Call NPICK
        Call CONFCK(ans)
        If ans = no  then  Call UBDCK
          else Call PART
        Go to LOOP
```

Subroutine RDATA
This initializes the entire procedure and reads in all AGVS information. The network data was stored in forward star form using a linked list structure. Set $Z_U = \infty$.

Subroutine PSP
This routine solves for the shortest path between input origin and destination over an input network using the partitioned shortest path procedure.

Subroutine FIPATH
This routine determines the nodes and node departure time in the shortest path found by the PSP subroutine.

Subroutine CONFCK
This routine checks for conflicts between the new AGV's active route and existing AGV traffic.

Subroutine PART
This routine partitions the set of paths represented by the lower bound solution evaluated into two sets of paths. The first set of paths represents the evaluated problem with the conflict arc removed and the second set of paths represents a resolution of the conflict by delaying the new AGV to resolve the detected conflict. Lower bounds are obtained for each partitioned set of paths and the bounds are placed on the active list.

Subroutine NPICK

This routine selects a lower bound solution from the active list for evaluation. Any method could be used, but this algorithm had a choice of two methods. One was LIFO, last in first out, and the second was Best-Bound, which checked each active problem and selected the problem with the best lower bound as the next problem to evaluate. If the active list is empty, the program terminates.

Subroutine UBCK

Compare this feasible solution with the best previously obtained solution. Retain the lowest solution as Z_U.

Worst Case Behavior

Computational complexity theory, as a practical tool for the investigation of combinatorial optimization problems, came into being around 1971 with the publication of two classical papers by S.A. Cook [14] and R.M. Karp [48]. These works laid the foundation for a technique used to establish the nondeterministic polynomial (NP) completeness of certain combinatorial problems. Such problems are unlikely to be solvable by an amount of computational effort which is bounded by a polynomial function of problem size. Worst case analysis provides valuable mathematical insights into the upper bound on computational effort and the relative solution difficulty of model types and individual algorithms. Polynomially bounded algorithms are referred to in the literature as "good" algorithms. Some shortest path algorithms are polynomially bounded, while others are exponentially bounded, yet some of the exponentially bounded algorithms consistently outperform the polynomially bounded algorithms.

Both the vehicle routing problem [58] and the zero-one integer programming problem [48] have been proven to be NP-hard. A reduction, a constructive transformation mapping an instance of this AGV routing problem to the zero-one integer programming problem, proves this problem is also NP-hard. Due to the difficulty of this NP-hard problem class, it is improbable that an optimal solution could be found in polynomial time. The solution method presented in this work utilizes a partitioned shortest path algorithm as a basis for the branch-and-bound procedure. Therefore any performance analysis of this procedure must be dependent on the performance of the shortest path algorithm.

The shortest path algorithm used in this work is a variant of an algorithm presented by Glover, Klingman, Phillips, and Schneider [39] called the partitioned shortest path algorithm. This particular algorithm has been shown to have computational complexity on the order of the number of nodes (N) times the number of arcs (A), $N \times A$. While this complexity is equal to the best time complexity found for any shortest path algorithms, the conflict resolution procedure presented in this work requires multiple calculations of shortest paths.

Consequently, the worst case behavior for the conflict resolution procedure would be equal to some multiple of $N \times A$, or would be measured by the number of partial problems which are decomposed and require shortest path computations. While each shortest path computation evaluates a slightly different network, an upper bound on this calculation is the total number of arcs in the original network.

For theoretical simplicity, the computational efficiency of a branch-and-bound algorithm, γ, is commonly measured by the number of partial problems, $P(\gamma)$, which are decomposed in the entire computation [43,51]. The actual time required to carry out the solution procedure is roughly given by $P(\gamma) \times \bar{t}$, where $\bar{t}$ is the average time required to solve a subproblem. In spite of the considerable success of branch-and-bound algorithms, it is empirically known that $P(\gamma)$ usually increases exponentially with the size of the given problem.

The NP-completeness argument is considered strong evidence for the nonexistence of a polynomial time branch-and-bound algorithm. Therefore, it seems unreasonable to expect that the number of partial problems evaluated, $P(\gamma)$, by any branch-and-bound algorithm can be made to grow less than exponentially. Ibaraki [45] proved that the mean number of partial problems which are decomposed in a branch-and-bound algorithm, grows at least as fast as exponentially with the problem size. As this AGV routing problem can be reduced to an integer programming problem, the worst case behavior would be expected to be exponential, $P(\gamma)\times\xi$, with $\xi \approx N\times A$.

The complexity of an algorithm has usually been measured by its worst-case behavior over all instances of the problem. The obvious problem with such a measure is that it gives little information about the usual or average performance of the algorithm. This is especially true of branch-and-bound procedures, which can have widely varying behaviors over all instances of a problem, The major limitation of worst case analysis is that it only provides a measure of worst case performance and does not normally provide information on the algorithm's average performance. Empirical tests have shown that algorithms with better worst case bounds do not always beat algorithms with better average performance. Consequently, it is felt average performance provides more useful information than a worst-case measure, provided the empirical tests are representative of "real" problems to be solved by the algorithm.

The example conflict resolution algorithm was tested extensively and the expected exponential time complexity can just begin to be seen with problems evaluating four shortest path problems, see Appendix A. Although experimental testing is not sufficient to provide worst case behavior, the results seem to validate all predictions stated earlier.

Average Behavior

The primary method used to evaluate an algorithm's computational complexity is empirical testing. The computer is used both to evaluate the efficiency of the algorithm and to provide statistics about the operations of key algorithmic steps under varying test conditions. Properly generated and utilized, these statistics allow researchers to both gain valuable insights into how different algorithms perform on a variety of problem topologies and to improve the design of specific algorithmic steps. An iterative modification, integration, and evaluation of key processes is directly analogous to the laboratory research of other disciplines.

Any evaluation procedure has its limitations. The major limitation of the empirical approach is that any extrapolation of the results to other problems must be considered speculative. The results are only valid for the computer and language used and the problems tested. The example algorithm was coded in FORTRAN and tested on a microcomputer (IBM XT *clone*) operating at 4.77 MHz. The algorithm was used to solve 141 test problems, with randomly generated nodes, arcs and existing traffic.

In order to test the effectiveness of the test algorithm's performance, 141 problems were generated and each problem solved twice. The first time a depth-first search strategy was used. This strategy always evaluates the last lower bound placed on the active list. In the event a conflict was discovered, the last lower bound represents a resolution of the conflict by delaying the new AGV to allow the conflicting AGV to pass. This strategy hopes to find a feasible solution early and then fathom subsequent sets of paths, resulting in the rapid conclusion of the algorithm. The second time a best-bound search strategy was used. This strategy evaluates all lower bound solutions on the active list and selects the smallest one for evaluation. This strategy speculates that the first feasible solution found will be the optimal solution, with all subsequent lower bounds fathomed.

The average performance of the algorithm was surprisingly good. In fact 100 of the 141 randomly generated problems required only one shortest path problem evaluation. This performance was exceptional considering a direct attempt was made to pack each network with AGVs. Table 1 presents the distribution of the number of problems for each number of nodes used in the random problem generation. The nonuniformity of the distribution of the number of test problems per nodes shows a majority of the problems had less than 40 nodes. This distribution was chosen to attempt to load the algorithm, to try put several AGVs on lower node count networks and thus obtain the expected exponential worst case performance of the test algorithm. The graphs presented in Appendix A validate the partial effectiveness of this attempt. The "hardest" problem consisted of 40 nodes and required the solving of four shortest path problems and the evaluation of eight total problem formulations.

Each test problem (with the exception of three problems) was randomly generated using a random network generator specially developed to generate bi-directional AGV networks. The generation of the test problem itself presented some interesting challenges as it was rather difficult to generate feasible test problems. The test network had to be connected and consist of a majority of bi-directional arcs, but also have the capability to include uni-directional arcs. The test network had to be realistic, for example a network having ten arcs incident to one node is nearly physically impossible to construct as an AGVS. Consequently for realism, node arc density was limited to six arcs and forced to average between two and four.

Table 1: Node Distribution for Test Problems.

Number of Nodes	Number of Problems	Number of Nodes	Number of Problems
7	1	72	1
10	19	74	1
15	6	75	2
17	2	76	1
20	32	77	1
25	3	78	3
27	1	80	5
30	7	83	1
40	9	85	2
45	3	88	1
47	1	90	2
50	5	91	2
51	1	92	1
55	1	94	1
59	1	95	3
60	6	96	1
61	1	97	1
62	1	98	3
70	2	99	2
71	1	100	4

Three of the networks tested represented actual AGV systems found in the literature [62]. Only three were used as there are not a lot of AGV systems pictorially represented in the literature. Existing AGV traffic was randomly generated for all problems, even for the three actual systems used. This is because the actual routing of AGVs in industrial systems appears to be a closely guarded corporate secret.

The random network generator began with the input of a number n, which represented the number of nodes in the network. A random number m based on n was generated which represented the total number of arcs for the network. A bi-directional arc counts as two directed arcs, but only count as one toward each node arc density. This number was generated to insure the node arc density did not exceed some maximum number, usually three, but occasionally four.

To randomly generate the arcs and insure a connected network was produced, a spanning tree consisting of *n-1* arcs, was first generated. To accomplish this two lists were maintained; one list consisted of connected nodes and the second list consisted of the unconnected nodes. For each generated arc two nodes were selected, one from each list. A check for duplication and a check for node arc density was made at each step. If either duplication existed, the entire arc was regenerated. Once the spanning tree was formed, other bi-directional arcs were generated up to some random proportion of *m*. When all bi-directional arcs were generated, uni-directional arcs were generated to complete the network.

A random number of existing AGVs was generated each with its own origin and destination. Each AGV's route was obtained using the PSP algorithm previously described. A check for conflicting traffic was performed and modifications to the routes were made, if required, to insure feasibility.

The Appendices provide graphical presentations of the algorithm's performance. These graphs pictorially verify the worst case performance predicted earlier, with the performance represented by the formula $P(\gamma)\times\bar{t}$. The first four graphs in Appendix A demonstrate that $P(\gamma)$ is more appropriately represented by the number of shortest path problems than by the total number of problems evaluated. One fact that may help explain this inconsistency is that all the network information is passed to the PSP algorithm by writing to and reading from a virtual disk. This process was more efficient than writing to a floppy disk drive, but not as efficient as using a hard disk (the test computer did not have a hard drive). These disk reads and writes are among the most time consuming micro-computer operations and tend to dominate the time needed to obtain optimal solutions. If a larger computer was used the disk reads and writes might be eliminated and the computation times reduced significantly. Appendix B provides the depth first performance and best bound performance of the algorithm on problems requiring the evaluation of only one shortest path problem. Appendix C provides the depth first performance and best bound performance of the algorithm on problems requiring the evaluation of two shortest path problems. These graphs depict the linear performance of the algorithm, given the number of shortest path problems solved.

Since $\mathfrak{t}$ was previously shown to be represented by N×A, it is not surprising that all the graphs in Appendix B and C using nodes, bi-directional arcs, and total arcs provide excellent correlation between their value and computation times. What might be considered surprising was the fact that all those predictors were better than the predictions using N×A. Table 2 presents regression equations predicting computation time using the depth first search = *DTime* and the best bound search = *BBTime* (Italicized words represent the shortened variable names used in Table 2). The equations were a result of stepwise, forward selection, and backward elimination procedures and evaluated the following variables: number of nodes = *Nodes*, number of bi-directional arcs = *BiArcs*, number of uni-directional arcs = *UniArcs*, number of total arcs = *TotArcs*, where *TotArcs* = 2×*BiArcs*+*UniArcs*, number of nodes times total arcs = *N*A*, and number of AGVs = *NAGV*. In Table 2 equations with an $R^2 < 99.0$ have only the best of the two search equations presented. All regression models agreed on the best model, and probably most surprising was the extreme accuracy the regression models had in predicting the computational performance of the algorithm. This author had never seen an R^2 value of 100.0 without the example being contrived.

With such superior regression results an attempt was made to combine the number of shortest path problems = *NSP* evaluated as an input variable to the equation. These results are provided as the last equations in Table 2.

Table 2: Regression Equations for Computational Performance.

One Shortest Path Evaluation

Regression Equation	R^2
DTime = 2.50 + .000483 N*A	96.0
DTime = .367 + .117 BiArcs	98.6
DTime = .243 + .132 Nodes	99.5
BBTime = .240 + .132 Nodes	99.4
DTime = .220 + .0525 TotArcs	100.0
BBTime = .216 + .0526 TotArcs	100.0
Two Shortest Paths Evaluation	
BBTime = 8.16 + .00112 N*A	96.4
BBTime = 1.88 + .280 BiArcs	98.1
DTime = 1.83 + .312 Nodes	99.2
BBTime = 1.78 + .310 Nodes	99.3
DTime = 1.98 + .125 TotArcs	99.9
BBTime = 1.93 + .125 TotArcs	99.8
Three Shortest Paths Evaluation	
DTime = 9.75 + .00184 N*A	92.2
DTime = 3.30 + .483 Nodes	99.1
BBTime = 3.19 + .483 Nodes	99.1
DTime = 2.21 + .467 BiArcs	99.1
BBTime = 2.09 + .467 BiArcs	99.2
DTime = 2.78 + .203 TotArcs	99.2
BBTime = 2.67 + .203 TotArcs	99.2
Combined Results	
DTime = -3.58 + 2.99 NSP + .0603 NSP*TA	99.2
BBTime = -3.48 + 2.90 NSP + .0600 NSP*TA	99.3
DTime = .725 - .0351 TotArcs + .0844 NSP*TA	99.5
BBTime = .708 - .0341 TotArcs + .0835 NSP*TA	99.5
DTime = -1.26+1.51 NSP+.0746 NSP*TA-.0222 TotArcs	99.8
BBTime = -1.23+1.47 NSP+.0739 NSP*TA-.0216 TotArcs	99.8

Chapter 7

CONCLUSIONS AND RECOMMENDATIONS

While AGVs are in use in many factories [5,29,30], organizations are not realizing the potential for productivity increases and vehicle cost savings resulting from bi-directional routing of their vehicles. As new system successes become better known, investments in new systems should increase significantly and existing systems should be modified to take advantage of this new routing control methodology.

The decrease in the cost of computing in general, and of micro- and mini-computers in particular, have made automated systems particularly economical. One may currently purchase a 640K micro-computer with 40 megabytes of hard disk storage, printer, and all systems software for under $3,000. If compatible bi-directional vehicle control software existed, even small organizations (if they could afford the AGV equipment) would be able to automate their material handling activities.

Unfortunately, as Egbelu and Tanchoco [24] indicated, the typical AGV system employs a uni-directional guide path network. They clearly demonstrated that the use of a bi-directional traffic flow network could lead to increased productivity if AGV control capability could be developed.

The primary aim of this research was the development of this control procedure for multiple AGVs on a complex bi-directional network. An example algorithm was developed, coded, and tested. The algorithm was the result of the combination of a modified partitioned shortest path procedure and the branch-and-bound problem solving methodology.

The basic structure of this procedure is similar to the structure of most traditional integer programming solution procedures using a branch-and bound framework. There are two primary ways the procedure developed here differs from traditional branch-and-bound procedures. The first difference is in the method used to solve the subproblems generated by the conflict resolution procedure. The generated subproblems for the AGV routing problem possess the structure of the shortest path problem. The partitioned shortest path algorithm provides efficient solutions to these subproblems and is a prime factor for the relative efficiency of the test algorithm developed in this work.

A second difference is in the way the additional constraints, implied by partitioning, are implemented. In the typical branch-and-bound based integer programming algorithms, the addition of a new constraint causes primal feasibility to be lost. While feasibility is lost with the addition of new constraints each subproblem is handled by one of two methods. The first method is a simple modification of the network by removing an arc from the previous network and resolving another shortest path problem. The second method can be done at the conflict detection step. The conflict is resolved by delaying the new AGV at a node until the conflicting traffic has passed. This second conflict resolution technique implies less work is done to attain a lower bound to these subproblems than with most branch-and-bound procedures. The advantage of this second method is validated by the average computational performance which was most accurately predicted by the number of shortest path problems evaluated and not the total number of problems evaluated.

The example algorithm which evolved from this research is capable of solving any AGV routing problem. The maximum size of a problem solvable with this procedure is limited only by the amount of computer memory resources available. Whatever memory is available, it should be used effectively, for instance use the efficient forward star representation of a network mentioned in this work.

Limitations of the Study

The major limitation of this study results from the nature of the problems tested. Since almost all of the problems were randomly generated, they may not be totally representative of problems found in actual practice. While the test algorithm was written to use a math coprocessor, the arc lengths in the problems were integer valued and most of the code used integer mode arithmetic. With a math coprocessor, the algorithm should perform well using real valued arc lengths. But, any extrapolation of the empirical results beyond the classes of problems considered and beyond the particular codes tested is speculative at best.

Another limitation is that the example algorithm was tested on only one computer and with only one compiler. Evaluation on other computers, preferably one currently used in AGV tracking, would provide a better indication of the ability for "real-time" AGV control. A larger computer, especially one with a hard disk, would most likely provide faster solution times. The use of a mini-computer could eliminate the required disk reads and writes and cause a significant reduction in the overall computation time of the algorithm.

Two search rules were tested in the example algorithm, the LIFO search strategy rule and the best bound search strategy. While significant computation time differences were not noted in the problems tested, it is certainly an area for future investigation. The total number of probl [illegible] evaluated did exhibit differences using each search strategy. However, since conflict delay resolutions can be resolved so quickly, reducing the total number of problems may not be as significant as initially thought. The randomly generated problems may not be a perfect indication of real AGV systems. As "real" AGV applications become available for testing, an evaluation of the LIFO and best bound search strategy may provide better information. Future research using actual AGV systems will provide more information on this control methodology's performance and real-time applicability.

Future research into an extension of this procedure should consider solving the *n* vehicle optimal routing problem. The procedure might attempt to apply the branch-and-bound methodology to all *n* vehicles, the problem initially considered. To begin, all vehicle's routes would be calculated using a shortest path procedure. A conflict detection procedure would find any conflicts that exist using these routings. If a conflict is detected, partition the set into five sets rather than the three sets obtained in this work. For example, consider AGV A and AGV B and set T^i. When a conflict is discovered generate the following five sets: T^{A2i} = { set of all paths from AGV A's origin to destination that do not use the detected conflict arc }, T^{A2i+1} = { set of all paths from AGV A's origin to destination that use the detected arc, but AGV A's time interval on the detected arc is outside of the time interval where the conflict arc was used by AGV B }, T^{B2i} = { set of all paths from AGV B's origin to destination that do not use the detected conflict arc }, T^{B2i+1} = { set of all paths from AGV B's origin to destination that use the detected arc, but AGV B's time interval on the detected arc is outside of the time interval where the conflict arc was used by AGV A }, and F = { set of all paths for each AGV which use the conflict arc during the conflicting time window identified }. This method appears to be a logical extension of the procedure presented in this work, and has obvious application to AGV system design.

As the procedure in this work and extensions of the procedure are applied to actual AGV systems, the improved AGVS performance should convince industry of the viability of bi-directional AGV routing. These concepts will continue the boom of AGVS applications and as control techniques continue to improve, the material handling industry will exhibit both increased profits and improved control.

BIBLIOGRAPHY

1. Agent, K.R. and Clark, J.D., "Evaluation of reversible lanes," Traffic Engineering and Control, Vol 23, No 11, November 1982, pp 551-555.

2. Allen, J.D., Helgason, R.V., and Kennington, J.L., "The Frequency Assignment Problem: A Solution via Nonlinear Programming," Technical Report 85-OR-5, Operations Research Department, Southern Methodist University, Dallas, Texas, June 1985.

3. Assad, A.A., "Multicommodity Network Flows - A Survey," Networks, Vol 8, No 1, Spring 1978, pp 37-91.

4. Baker, E.K., "An Exact Algorithm for the Time-Constrained Traveling Salesman Problem," Operations Research, Vol 31, No 5, September-October 1983, pp 938-945.

5. Beck, L., "Lift trucks and AGVs speed warehouse operations," Modern Materials Handling, Vol 40, No 12, October 1985, pp 76-78.

6. Bellmore, M., Bennington, G., and Lubore, S., "A Multivehicle Tanker Scheduling Problem," Transportation Science, Vol 5, No 1, February 1971, pp 36-47.

7. Billheimer, J.W. and Gray, P., "Network Design with Fixed and Variable Cost Elements," Transportation Science, Vol 7, No 1, February 1973, pp 49-74.

8. Bodin L.D. and Golden B.L., "Classification in Vehicle Routing and Scheduling," Networks, Vol 11, No 2, Summer 1981, pp 97-108.

9. Bodin L.D., Golden B.L., Assad, A.A., and Ball, M.O., "Routing and Scheduling of Vehicles and Crews: The State of the Art," Computers and Operations Research, Vol 10, No 2, 1983, pp 63-212.

10. Bodin L.D. and Kursh, S.J., "A Computer Assisted System for the Routing and Scheduling of Street Sweepers," Operations Research, Vol 26, No 4, July-August 1978, pp 525-537.

11. Boldrin, B., "Applications and Procurement of Automated Guide Vehicle Systems," Robotics Engineering, Vol 8, No 2, February 1986, pp 10-14.

12. Clarke, G. and Wright, J.W., "Scheduling of Vehicles from a Central Depot to a Number of Delivery Points," Operations Research, Vol 12, No 4, July-August 1964, pp 568-581.

13. Considerations for Planning and Installing Automatic Guided Vehicle Systems, Material Handling Institute, 1983.

14. Cook, S.A., "The complexity of theorem-proving procedures," Proceedings of 3rd Annual ACM Symposium on Theory of Computing, Association for Computing Machinery, New York, 1971, pp 151-158.

15. Cyrus, J.P. and Kusiak, A., "S[illegible]duling Problems in Automated Guided Vehicle Systems," Technical Repor[illegible]/01/84, Department of Industrial Engineering, Technical University of Nova Scotia, Halifax, Nova Scotia, August 1984.

16. Cyrus, J.P. and Kusiak, A., "The Vehicle Scheduling Problem with Time Windows in Automated Guided Vehicle Systems," Working Paper #11/84, Department of Industrial Engineering, Technical University of Nova Scotia, Halifax, Nova Scotia, October 1984.

17. Dantzig, G.B. and Fulkerson, D.R., "Minimizing the Number of Tankers to Meet a Fixed Schedule," Naval Research Logistics Quarterly, Vol 1, 1954, pp 217-222.

18. Davies, P., "Automatic vehicle classification techniques: lane, speed and basic type classification," Traffic Engineering and Control, Vol 24, No 4, April 1983, pp 195-201.

19. Deo, N. and Pang, C., "Shortest Path Algorithms: Taxonomy and Annotation," Networks, Vol 14, No 2, Summer 1984, pp 275-323.

20. Dial, R., Glover, F., Karney, D., and Klingman, D., "A Computational Analysis of Alternative Algorithms and Labeling Techniques for Finding Shortest Path Trees," Networks, Vol 9, No 3, Fall 1979, pp 215-248.

21. Dreyfus, S.E., "An Appraisal of Some Shortest Path Algorithms," Operations Research, Vol 17, No 3, May-June 1969, pp 395-412.

22. Egbelu, P.J. and Tanchoco, J.M.A., "Operational Considerations for the Design of Automatic Guided Vehicle Based Material Handling Systems," Technical Report No 8201, Department of Industrial Engineering and Operations Research, Virginia Polytechnic Institute and State University, Blacksburg, Virginia, 1982.

23. Egbelu, P.J. and Tanchoco, J.M.A., "Characterization of Automatic Guided Vehicle Dispatching Rules," International Journal of Production Research, Vol 22, No 3, 1984, pp 359-374.

24. Egbelu, P.J. and Tanchoco, J.M.A., "Potentials for Bi-Directional Guide-Path for Automated Guided Vehicle Based Systems," International Journal of Production Research, Vol 24, No 5, September/October 1986, pp 1075-1097.

25. Enscore, E.E., Pegden, C.D., and Cavalier, T.M., "A Math Programming Formulation of the Meet/Pass Problem," Unpublished Research Proposal, Department of Industrial Engineering, Pennsylvania State University, University Park, Pennsylvania, 1984.

26. Fisher, M. and Jaikumar, R. "A Generalized Assignment Heuristic for Vehicle Routing," Networks, Vol 11, No 2, Summer 1981, pp 109-124.

27. Fitzgerald, K.R., "Assembly and stacking highlight AGVS trends," Modern Materials Handling, Vol 40, No 10, September 1985, pp 92-96.

28. Fitzgerald, K.R., "How to estimate the number of AGVs you need," Modern Materials Handling, Vol 40, No 12, October 1985, p 79.

29. Fitzgerald, K.R., "Our AGVs routinely leave the guidepath," Modern Materials Handling, Vol 41, No 1, January 1986, pp 80-81.

30. Fitzgerald, K.R., "Ford unveils a stunning example of integration," Modern Materials Handling, Vol 41, No 7, June 1986, pp 64-70.

31. Frank, O., "Two-Way Traffic on a Single Line of Railway," Operations Research, Vol 14, No 5, September-October 1966, pp 801-811.

32. Fujii, S. and Sandoh, H., "A Routing Algorithm for Automated Guided Vehicles in FMS," Proceedings of the IXth International Conference on Production Research, Vol II, August 1987, pp 2261-2267.

33. Garey, M.R., Graham, R.L., and Johnson, D.S., "Performance Guarantees for Scheduling Algorithms," Operations Research, Vol 26, No 1, January-February 1978, pp 3-21.

34. Geoffrion, A.M. and Marsten, R.E., "Integer Programming Algorithms: A Framework and State-of-the-Art Survey," Management Science, Vol 18, No 9, May 1972, pp 465-491.

35. Gillett, B.E. and Johnson, J.G., "Multi-terminal Vehicle-Dispatch Algorithm," OMEGA, The International Journal of Management Science, Vol 4, No 6, 1976, pp 711-718.

36. Gillett, B.E. and Miller, L.R., "Heuristic Algorithm for the Vehicle-Dispatch Problem," Operations Research, Vol 22, No 2, March-April, 1974, pp 340-349.

37. Glover, F., Glover, R., and Klingman, D.D., "Computational Study of an Improved Shortest Path Algorithm," Networks, Vol 14, No 1, Spring 1984, pp 25-36.

38. Glover, F., Klingman, D.D., and Phillips, N.V., "A New Polynomially Bounded Shortest Path Algorithm," Operations Research, Vol 33, No 1, January-February 1985, pp 65-73.

39. Glover, F., Klingman, D.D., Phillips, N.V., and Schneider, R.F., "New Polynomial Shortest Path Algorithms and their Computational Attributes," Management Science, Vol 31, No 9, September 1985, pp 1106-1128.

40. Golden, B.L. and Wong, R.T., "Capacitated Arc Routing Problems," Networks, Vol 11, No 3, Fall 1981, pp 305-315.

41. Gormley, T.W., "A Philosophy of Control ...," Proceedings of the 5th International Conference on Automation in Warehousing, White, J.A. (ed), December 1983, IFS (Publications) Ltd and North-Holland, Bedford England.

42. Gould, L., Private communication, October 24, 1986, Modern Material Handling, Associate Editor, Newton, Massachusetts.

43. Ibaraki, T., "Theoretical Comparisons of Search Strategies in Branch-and-Bound Algorithms," International Journal of Computer and Information Sciences, Vol 5, No 4, December 1976, pp 315-344.

44. Ibaraki, T., "Computational Efficiency of Approximate Branch-and-Bound Algorithms," Mathematics of Operations Research, Vol 1, No 3, August 1976, pp 287-298.

45. Ibaraki, T., "On the Computational Efficiency of Branch-and-Bound Algorithms," Journal of Operations Research Society of Japan, Vol 20, No 1, March 1977, pp 16-35.

46. Ibaraki, T., "The Power of Dominance Relations in Branch-and-Bound Algorithms," Journal of the Association for Computing Machinery, Vol 24, No 2, April 1977, pp 264-279.

47. Ibaraki, T., "Depth-m Search in Branch-and-Bound Algorithms," International Journal of Computer and Information Sciences, Vol 7, No 4, December 1978, pp 315-343.

48. Karp, R.M., "Reducibility among combinatorial problems," in R.E. Miller and J.W. Thatcher (eds), Complexity of Computer Computations, Plenum Press, New York, 1972, pp 85-103.

49. Kennington, J.L., "A Survey of Linear Cost Multicommodity Network Flows," Operations Research, Vol 26, No 2, March-April 1978, pp 209-236.

50. Koff, G.A., "Basics of AGVS," SME ULTRATECH, Vol 1, Conference Proceedings September 1986, pp 3.1-3.17.

51. Kohler, W.H. and Steiglitz, K., "Characterization and Theoretical Comparison of Branch-and-Bound Algorithms for Permutation Problems," Journal of the Association for Computing Machinery, Vol 21, No 1, January 1974, pp 140-156.

52. Kulkarni, R.V. and Bhave, P.R., "Integer programming formulations of vehicle routing problems," European Journal of Operations Research, Vol 20, No 1, April 1985, pp 58-67.

53. Kulwiec, R., "Automatic Guided Vehicle Systems," Plant Engineering, Vol 36, No 1, 7 January 1982, pp 50-58.

54. Kulwiec, R., "Trends in Automatic Guided Vehicle Systems," Plant Engineering, Vol 38, No 23, 11 October 1984, pp 66-73.

55. Land, A.H., and Doig, A.G., "An Automatic Method of Solving Discrete Programming Problems," Econometrica, Vol 28, No 3, July 1960, pp 497-520.

56. Lasecki, R.R., "AGV System Selection Methodology," SME ULTRATECH, Vol 1, Conference Proceeding September 1986, pp 3.81-3.96.

57. Lasecki, R.R., "AGVs: The Latest in Material Handling Technology," CIM Technology, Vol 5, No 4, Winter 1986, pp 90-94.

58. Lenstra, J. and Kan, A.R. "Complexity of Vehicle Routing and Scheduling Problems," Networks, Vol 11, No 2, Summer 1981, pp 221-227.

59. Levin, A., "Scheduling and Fleet Routing Models for Transportation Systems," Transportation Science, Vol 5, No 3, August 1971, pp 232-255.

60. Lin, S., and Kernigham, B., "An Effective Heuristic Algorithm for the Traveling Salesman Problem," Operations Research, Vol 21, No 2, March-April 1973, pp 498-516.

61. Maxwell, W.L., "Solving Material Handling Design Problems With OR," Industrial Engineering, Vol 13, No 4, April 1981, pp 58-69.

62. Maxwell, W.L. and Muckstadt, J.A., "Design of Automatic Guided Vehicle Systems," IIE Transactions, Vol 14, No 2, June 1982, pp 114-124.

63. Miller, R.K., "AGVS: A Needed Technology," Automated Guided Vehicle Systems Vol 1: Technology and Application, SEAI Technical Publications, Madison, Georgia, 1985, pp 1-11.

64. Newton, D., "Simulation Model Calculates How Many Automated Guided Vehicles Are Needed," Industrial Engineering, Vol 17, No 2, February 1985, pp 68-78.

65. Orloff, C.S., "Route Constrained Fleet Scheduling," Transportation Science, Vol 10, No 2, May 1976, pp 149-168.

66. Orloff, C. and Caprera, D., "Reduction and Solution of Large Scale Vehicle Routing Problems," Transportation Science, Vol 10, No 4, November 1976, pp 361-373.

67. Petersen, E.R., "Over-the-road Transit Time for a Single Track Railway," Transportation Science, Vol 8, No 1, February 1974, pp 65-74.

68. Phillips, D.T., "Simulation of Material Handling Systems: When and Which Methodology," Industrial Engineering, Vol 12, No 9, September 1980, pp 65-77.

69. Quinn, E.B., "A simulation based system for automatic development and testing of AGV control software," Proceedings of the 2nd International Conference on Automated Guided Vehicle Systems, Warnecke, H.J. (ed), June 1983, North Holland, Oxford England.

70. Schwind, G.F., "AGVS for assembly: flexible layout, easy expansion," Material Handling Engineering, Vol 40, No 10, September 1985, pp 58-62.

71. Schwind, G.F., Private communication, October 27, 1986, Material Handling Engineering, Executive Editor, Cleveland, Ohio.

72. Schwind, G.F., Private communication, November 3, 1986, Material Handling Engineering, Executive Editor, Cleveland, Ohio.

73. Shier, D.R., "On Algorithms for Finding the k Shortest Paths in a Network," Networks, Vol 9, No 3, Fall 1979, pp 195-214.

74. Smith, D.R., "Random Trees and the Analysis of Branch-and-Bound Procedures," Journal of the Association for Computing Machinery, Vol 31, No 1, January 1984, pp 163-188.

75. Solomon, M.M., "Vehicle Routing and Scheduling with Time Window Constraints: Models and Algorithms," Working paper 83-12-02, Department of Decision Sciences, The Wharton School, University of Pennsylvania, Philadelphia, Pennsylvania, December 1983.

76. Solomon, M.M., "On the Worst-Case Performance of some Heuristics for the Vehicle Routing and Scheduling Problem with Time Window Constraints", Working paper 83-12-03, Department of Decision Sciences, The Wharton School, University of Pennsylvania, Philadelphia, Pennsylvania, January 1984.

77. Spinelli, J.J., "The Effects of Load/Unload Times and Network Zoning on an Automated Guided Vehicle System," Unpublished Masters Paper, Department of Industrial and Management Systems Engineering, The Pennsylvania State University, University Park, Pennsylvania, November 1987.

78. Tarjan, R.E., "A Simple Version of Karzanov's Blocking Flow Algorithm," Operations Research Letters, Vol 2, No 6, March 1984, pp 265-268.

79. Weingarten, C., "The One-way Preference Street," Traffic Engineering, Vol 28, No 6, March 1958, pp 21-22.

80. Wenzel, C.D., Private communication, October 13, 1986, SPS Technologies, Automated Systems Division, Hatfield, Pennsylvania.

81. Young, E.L., Private communication, November 14, 1986, Litton Industrial Automation Systems Inc., Automated Vehicles Systems, Zeeland, Michigan.

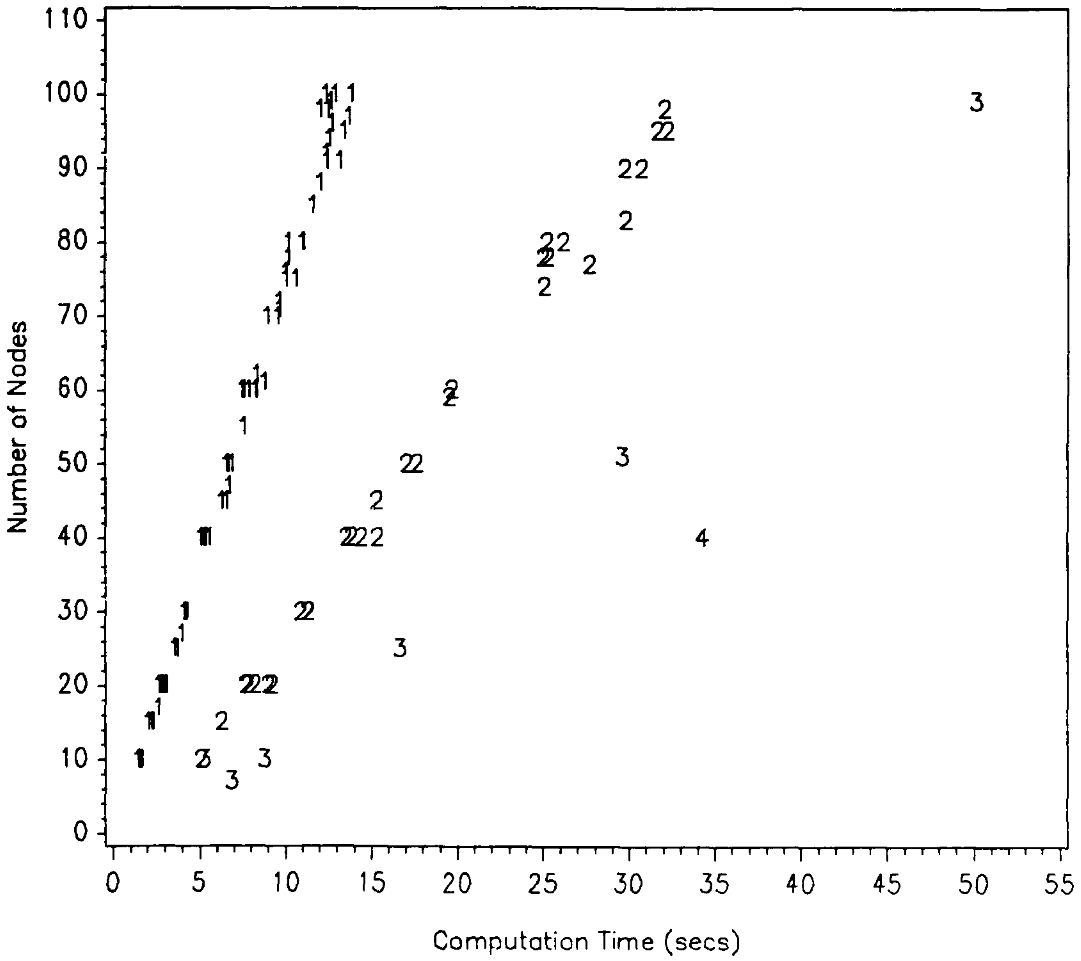
Appendix A

GENERAL GRAPHS OF TEST ALGORITHM'S PERFORMANCE

Depth First Time Performance

vs.

Number of Network Nodes

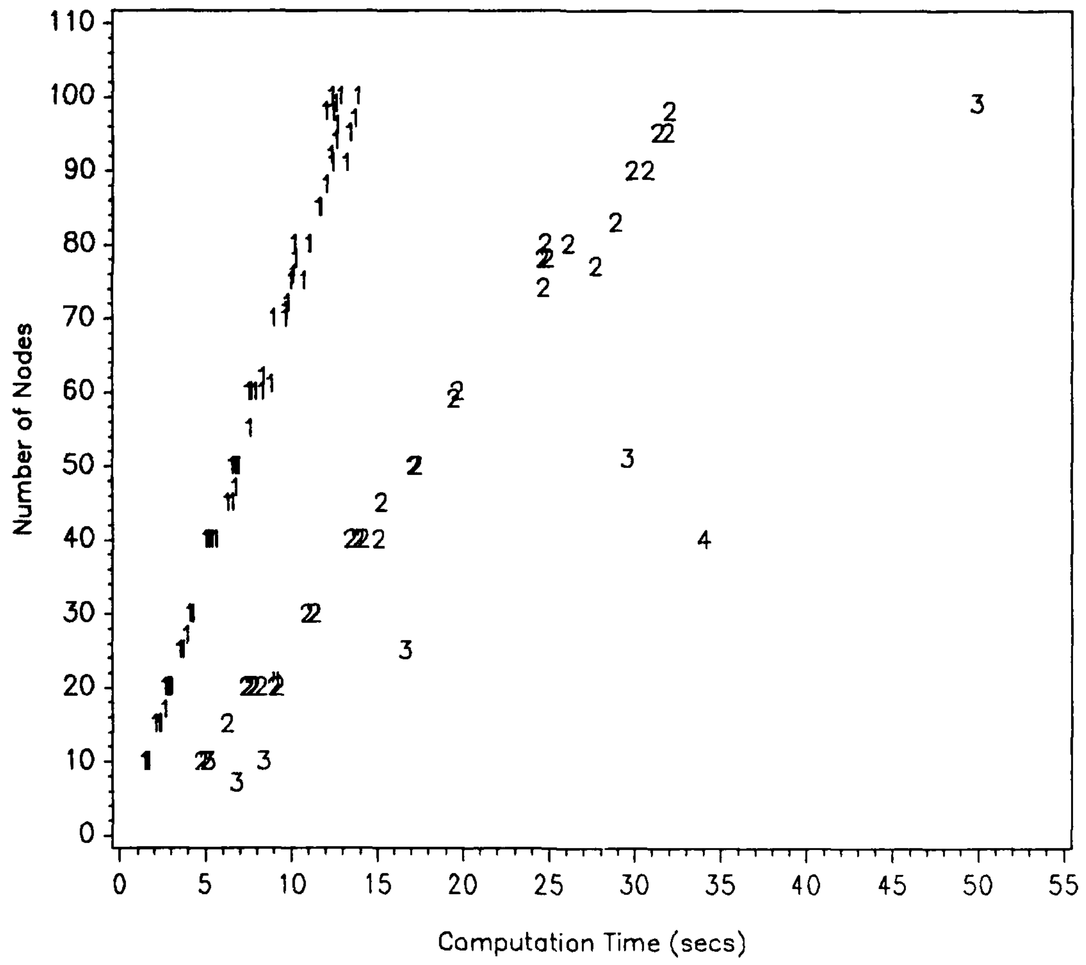


The Plotted Number represents the number of shortest path problems solved

Best Bound Time Performance

vs.

Number of Network Nodes

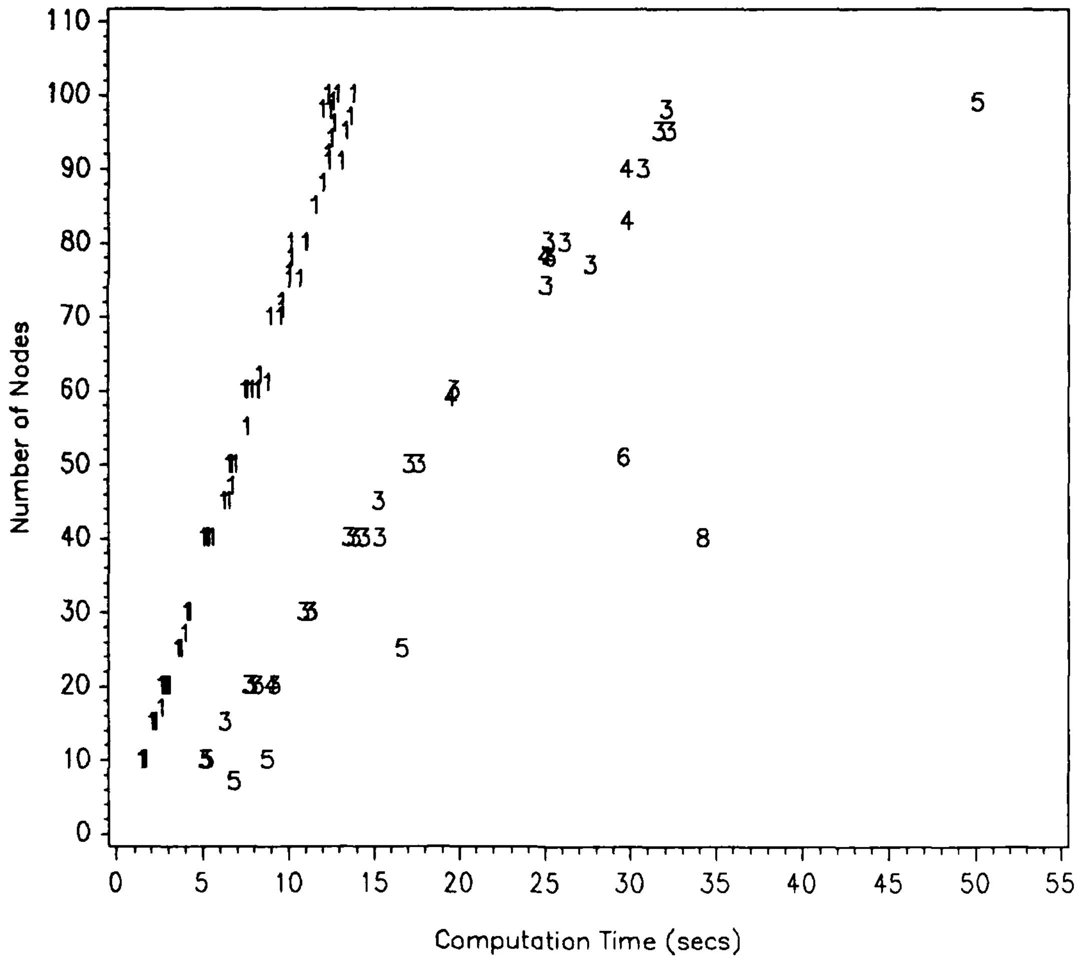


The Plotted Number represents the number of shortest path problems solved

Depth First Time Performance

vs.

Number of Network Nodes

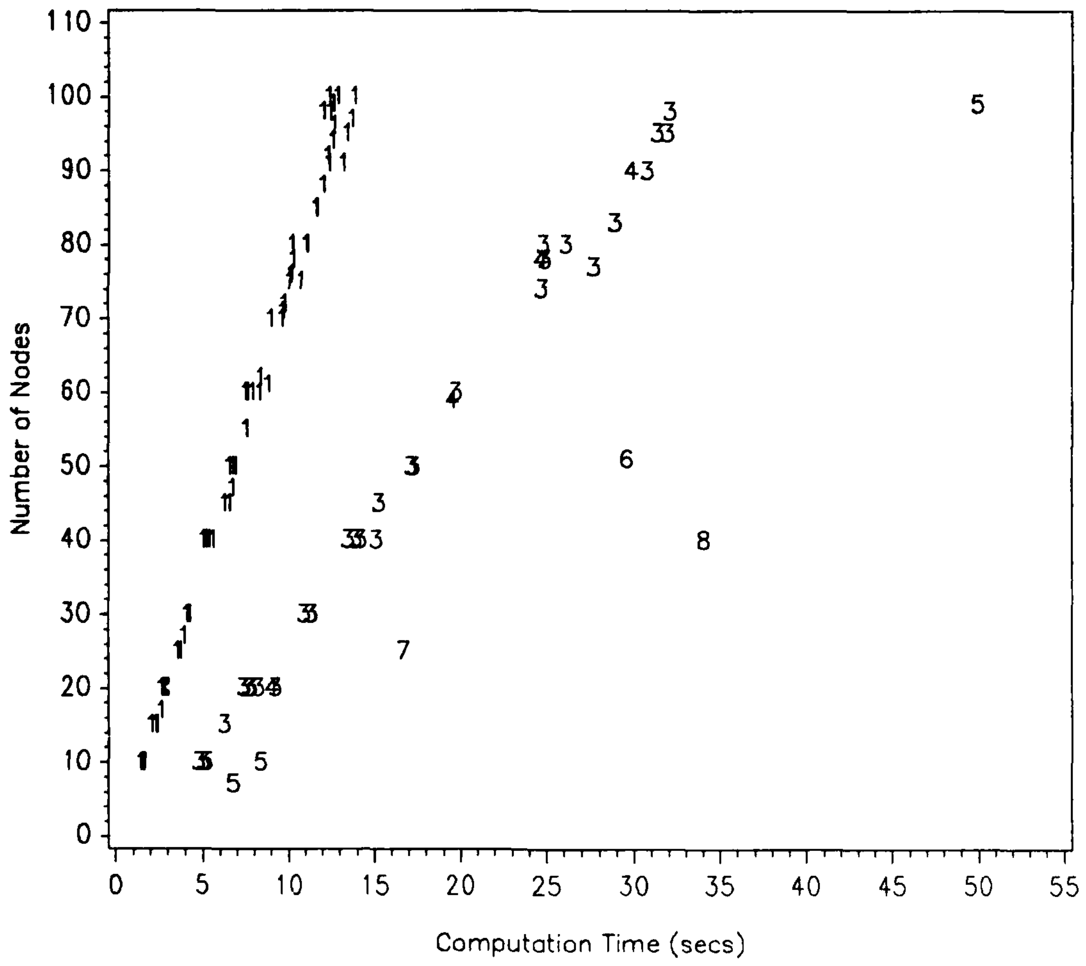


The Plotted Number represents the total number of problems solved

Best Bound Time Performance

vs.

Number of Network Nodes

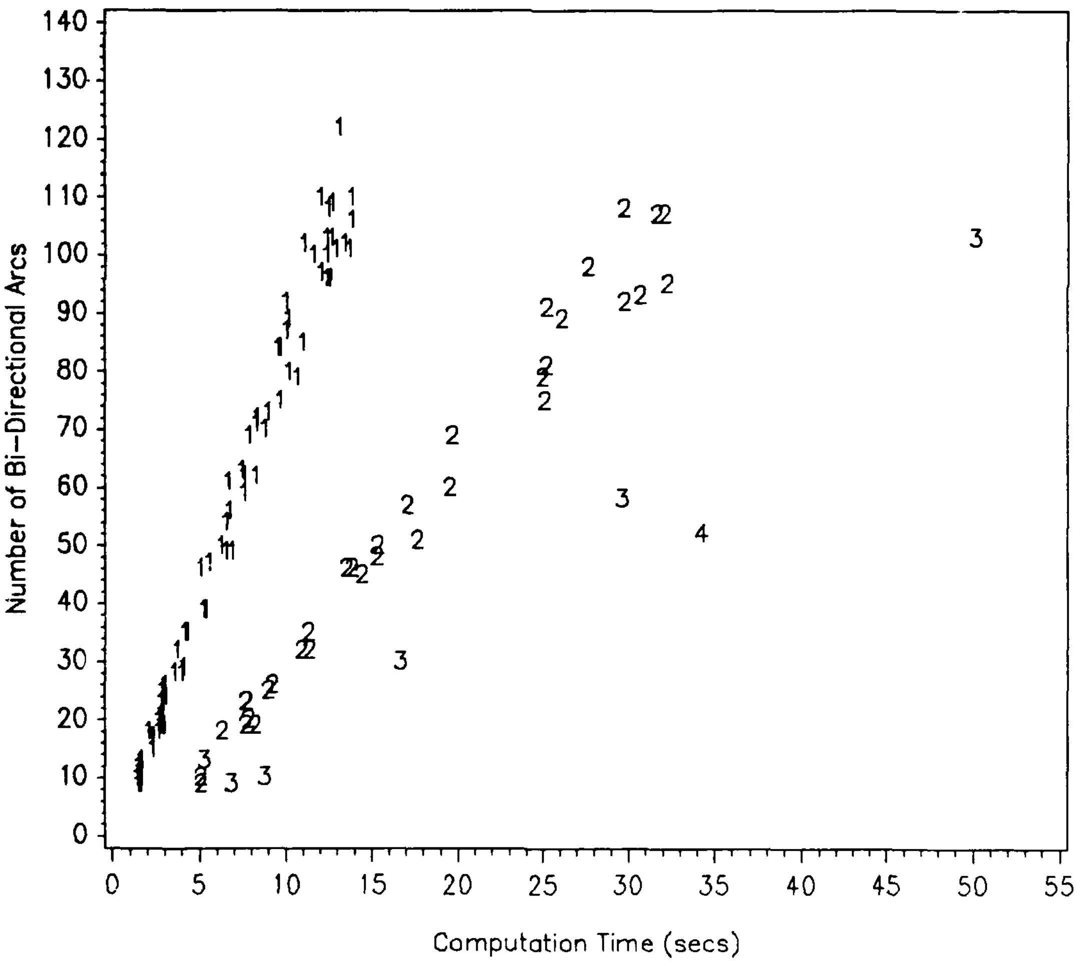


The Plotted Number represents the total number of problems solved

Depth First Time Performance

vs.

Number of Bi-directional Arcs

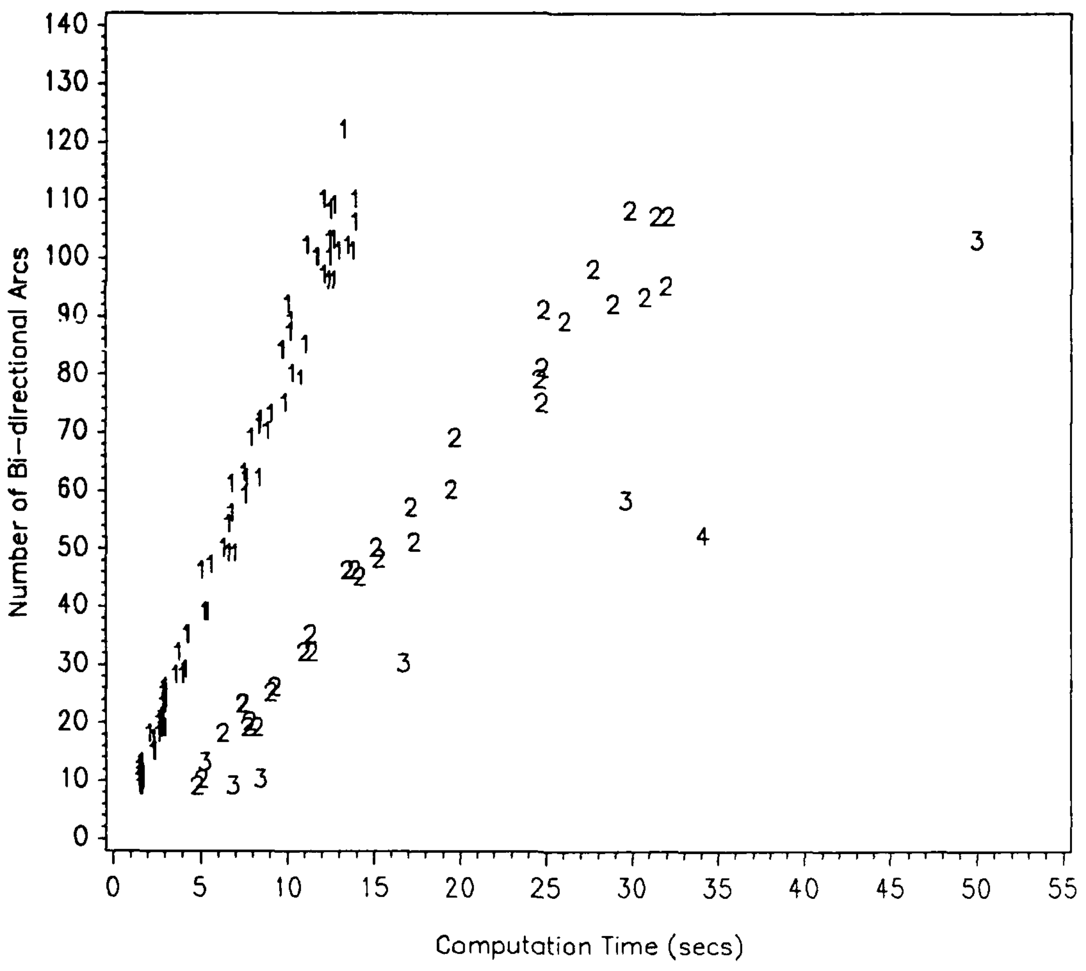


The Plotted Number represents the number of shortest path problems solved

Best Bound Time Performance

vs.

Number of Bi-directional Arcs

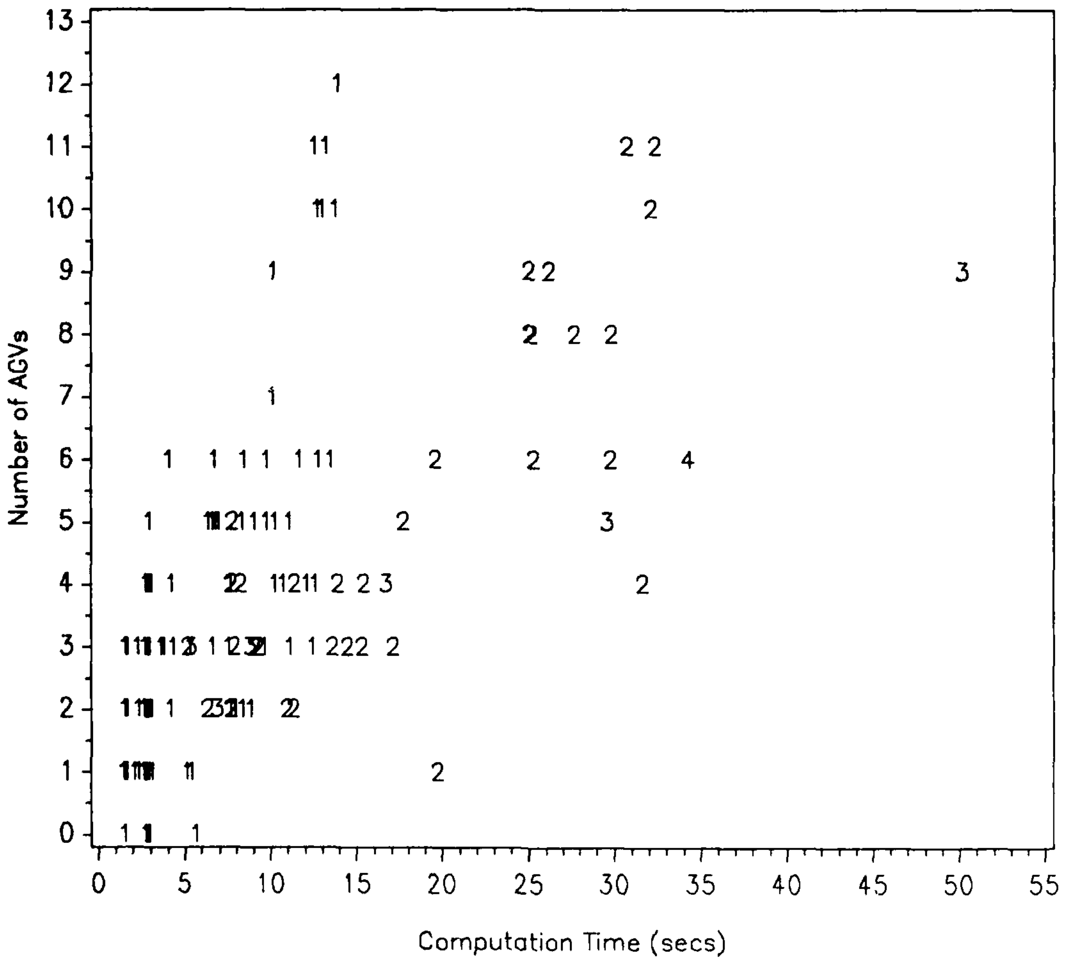


The Plotted Number represents the number of shortest path problems solved

Depth First Time Performance

vs.

Number of AGVs

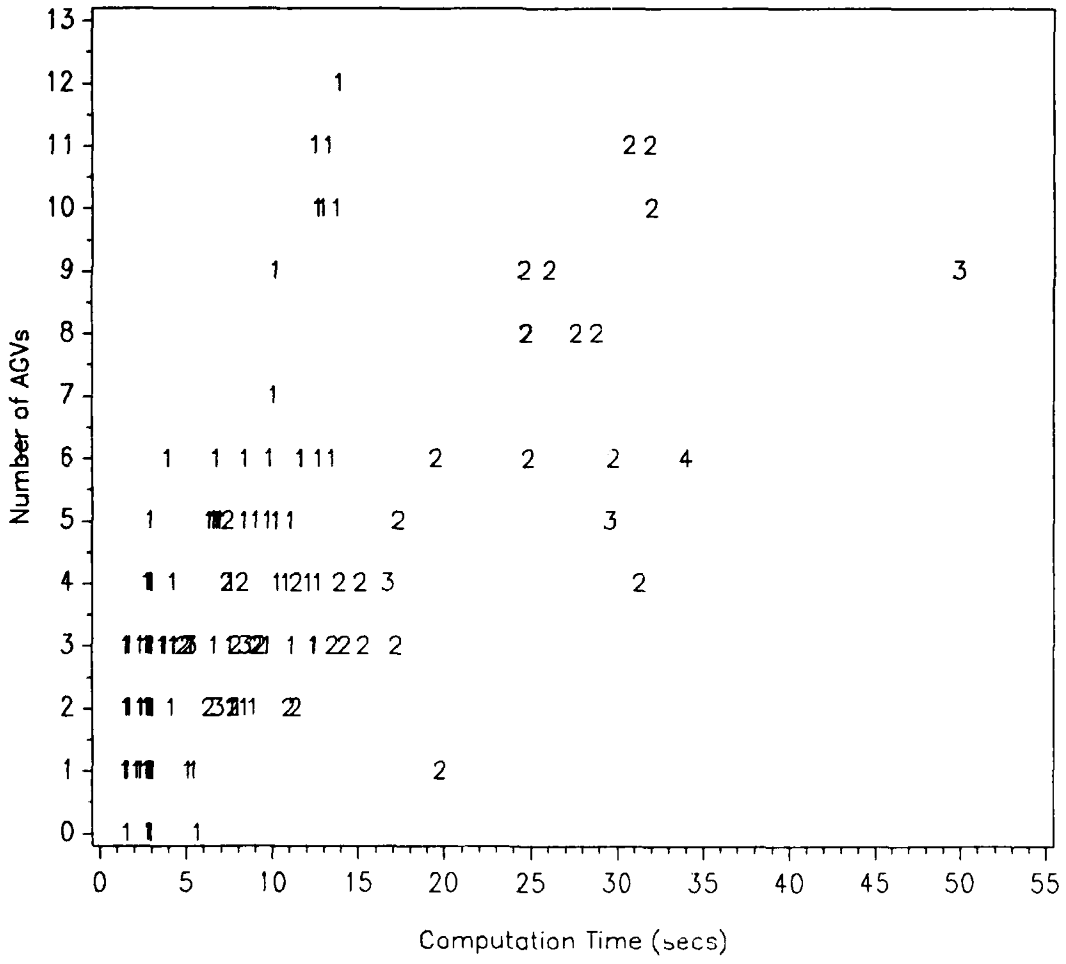


The Plotted Number represents the number of shortest path problems solved

Best Bound Time Performance

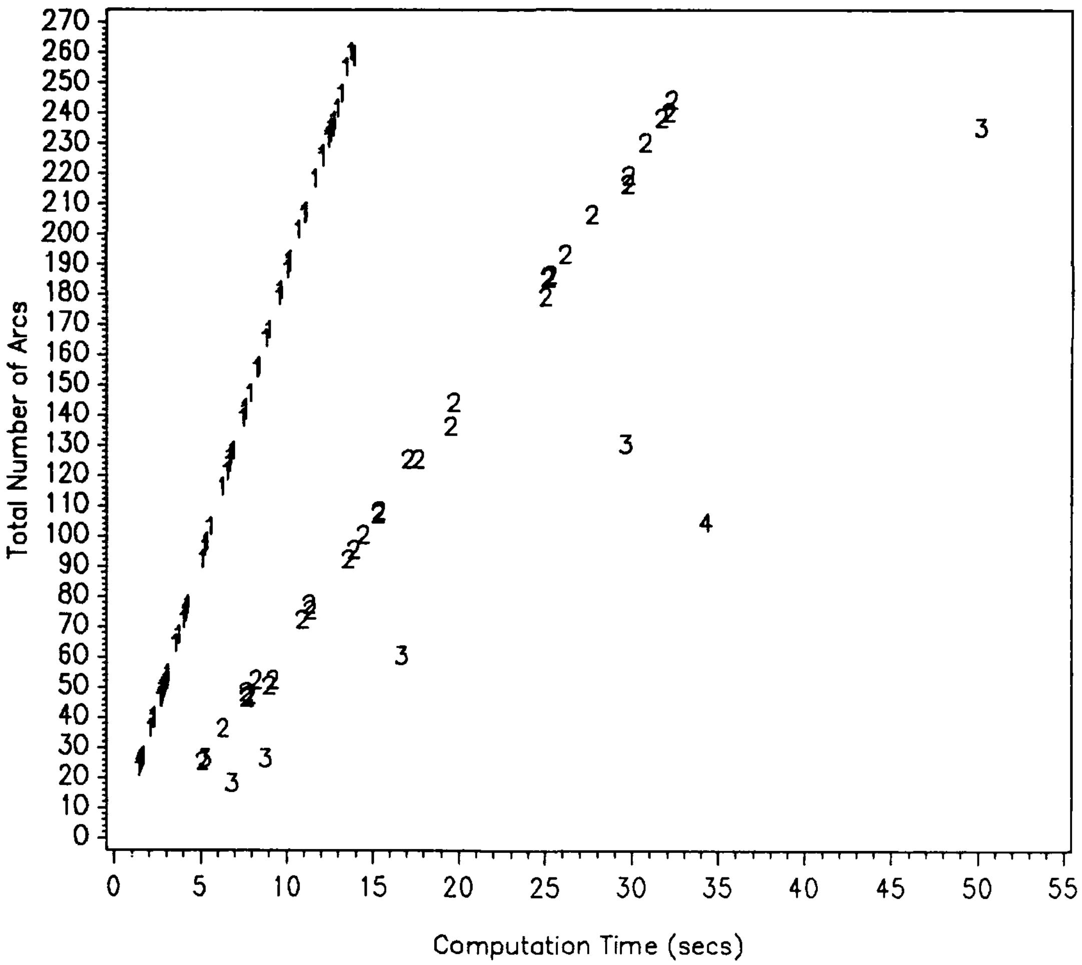
vs.

Number of AGVs



The Plotted Number represents the number of shortest path problems solved

Depth First Time Performance
vs.
Total Number of Arcs

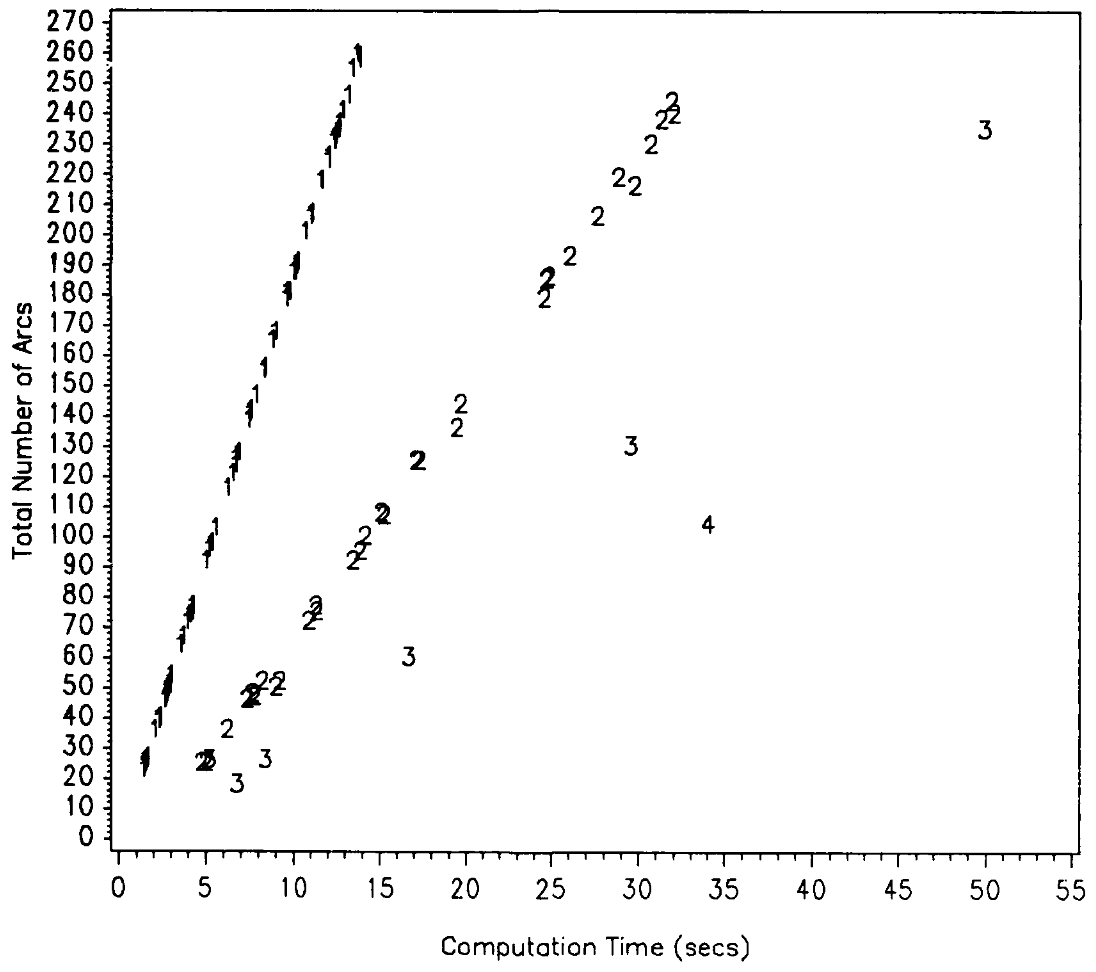


The Plotted Number represents the number of shortest path problems solved

Best Bound Time Performance

vs.

Total Number of Arcs



The Plotted Number represents the number of shortest path problems solved

Appendix B

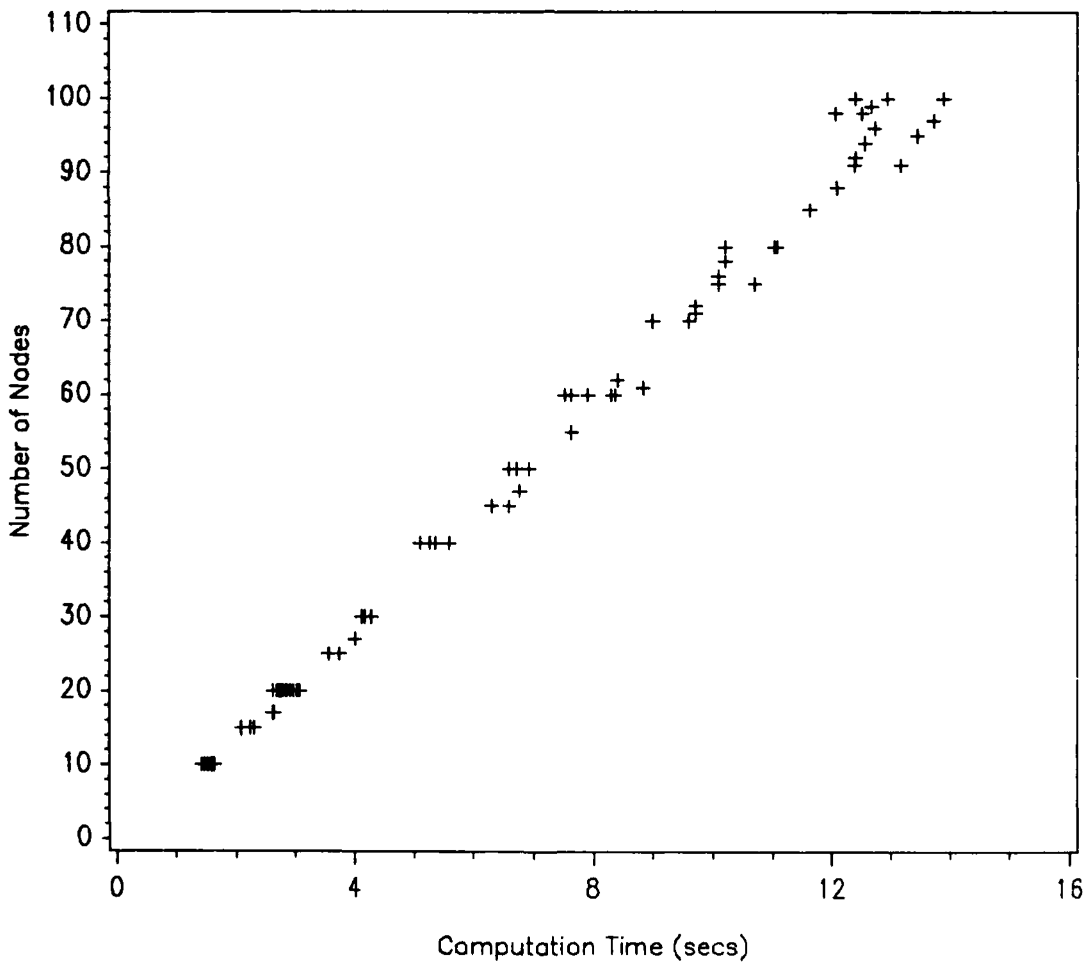
ONE SHORTEST PATH PROBLEM PERFORMANCE GRAPHS

One Shortest Path Evaluated

Depth First Time Performance

vs.

Number of Network Nodes



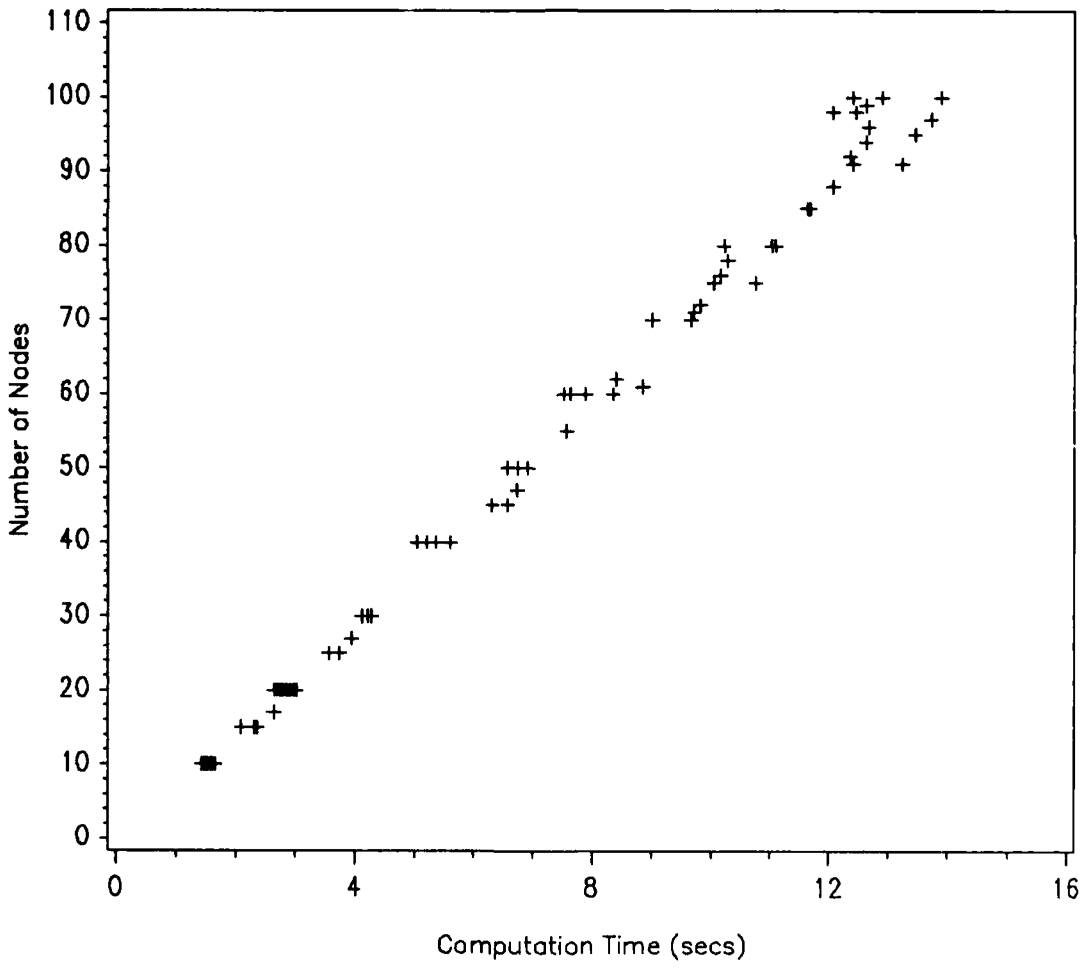
DTime = .243 + .132 * #Nodes R–Sq = 99.5

One Shortest Path Evaluated

Best Bound Time Performance

vs.

Number of Network Nodes



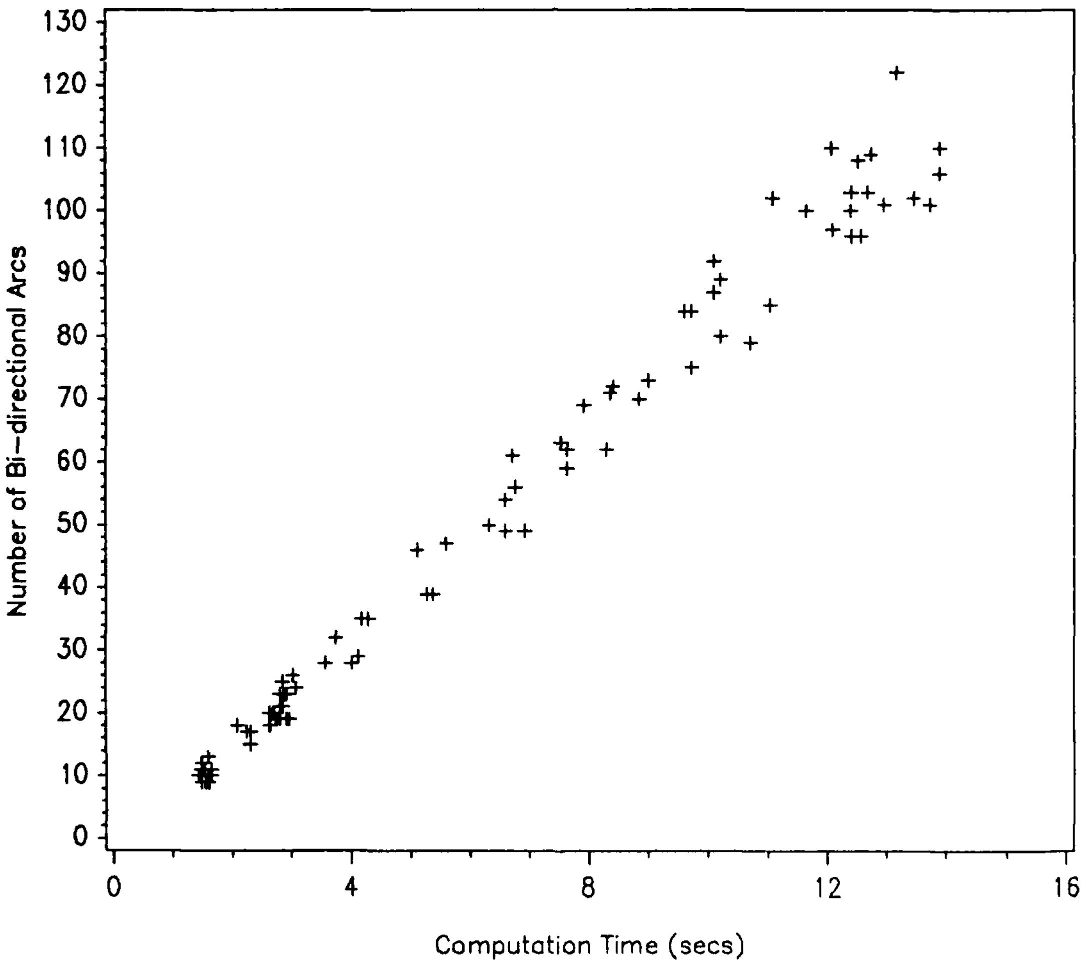
BBTime = .240 + .132 * #Nodes R–Sq = 99.4

One Shortest Path Evaluated

Depth First Time Performance

vs.

Number of Bi-directional Arcs



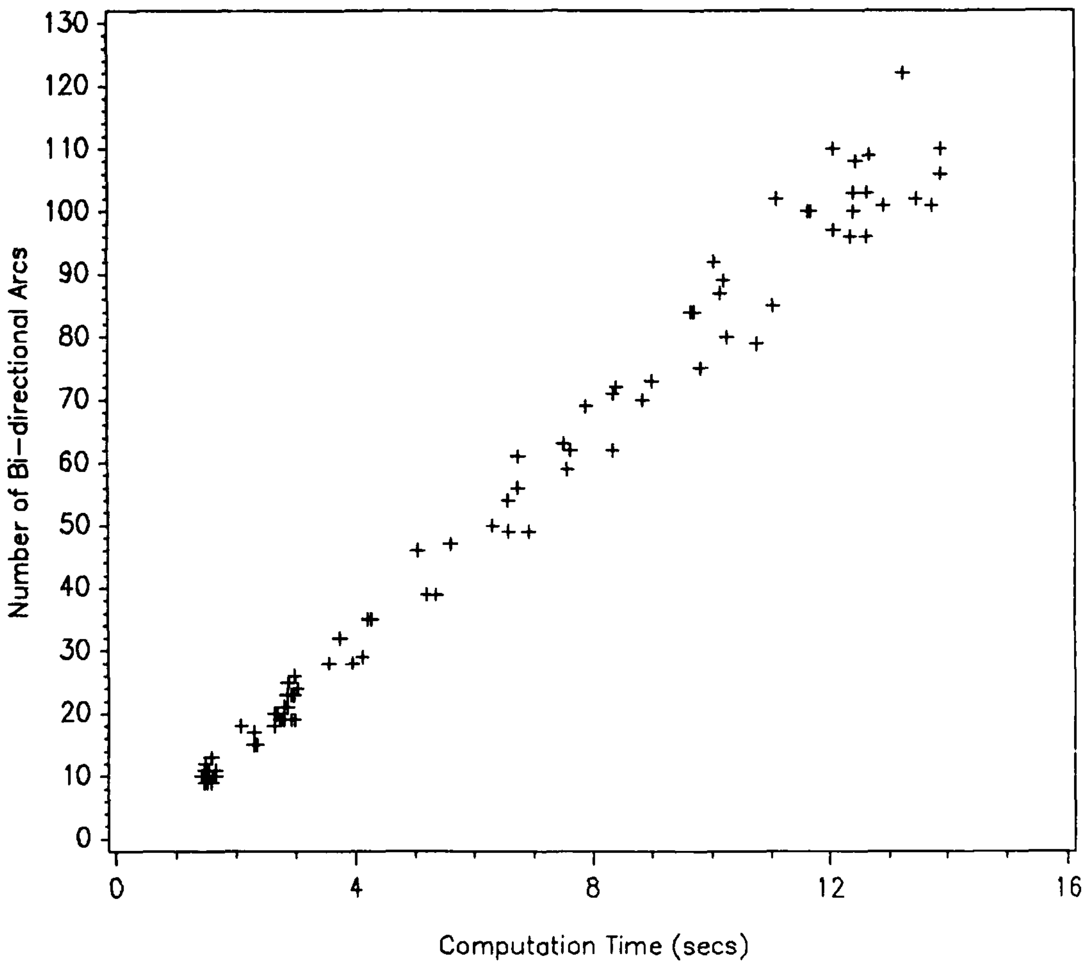
DTime = .367 + .117 * #BiArcs R–Sq = 98.6

One Shortest Path Evaluated

Best Bound Time Performance

vs.

Number of Bi-directional Arcs



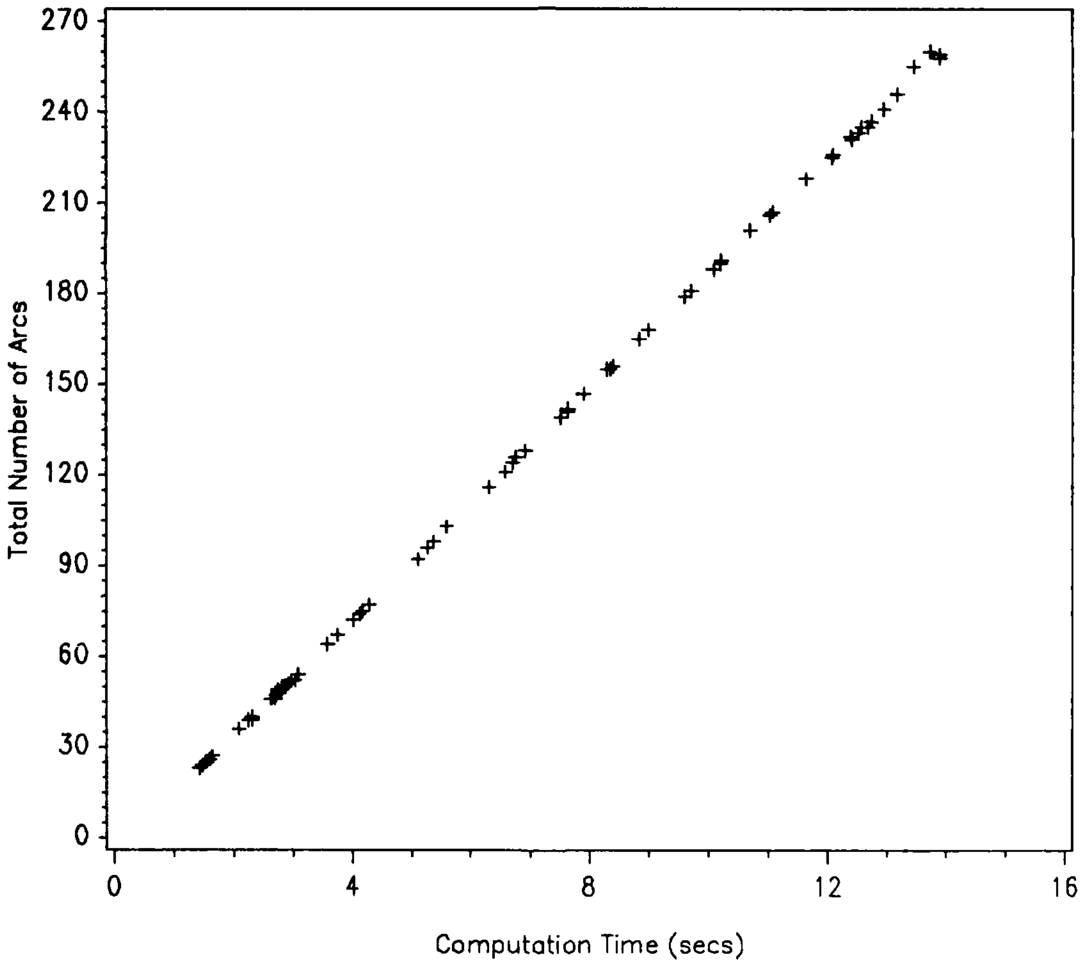
BBTime = .363 + .117 * #BiArcs R–Sq = 98.6

One Shortest Path Evaluated

Depth First Time Performance

vs.

Total Number of Arcs



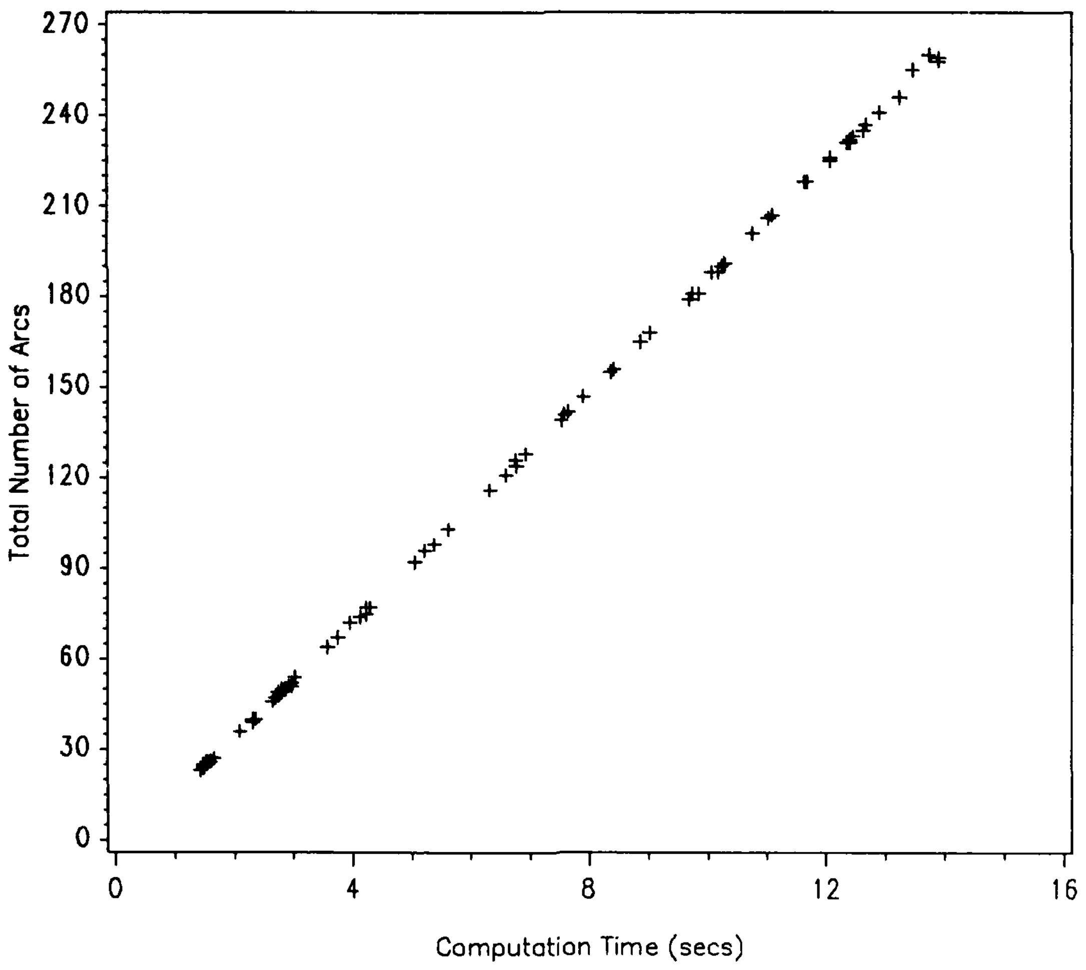
DTime = .220 + .0525 * #TotArcs R−Sq = 100.0

One Shortest Path Evaluated

Best Bound Time Performance

vs.

Total Number of Arcs



BBTime = .216 +.0526 * #TotArcs R−Sq = 100.0

Appendix C

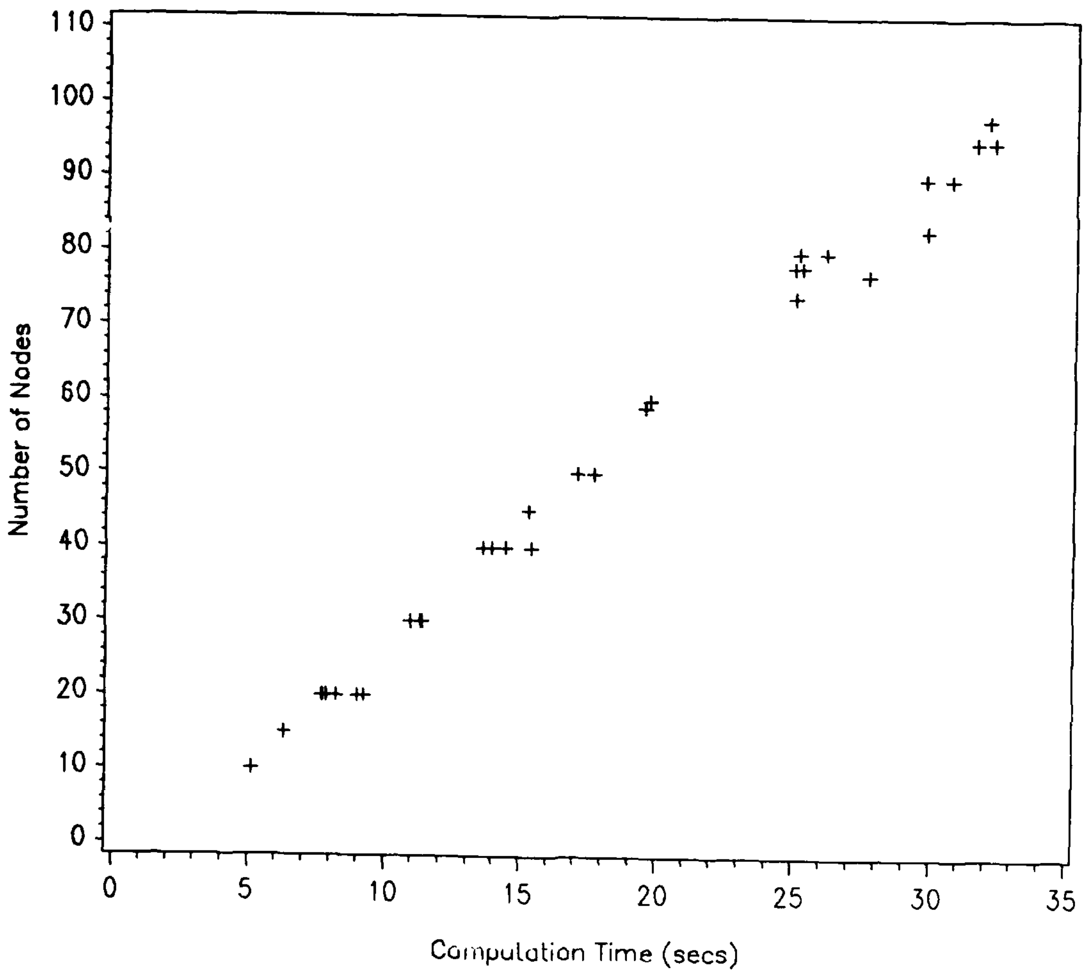
TWO SHORTEST PATH PROBLEMS PERFORMANCE GRAPHS

Two Shortest Paths Evaluated

Depth First Time Performance

vs.

Number of Network Nodes



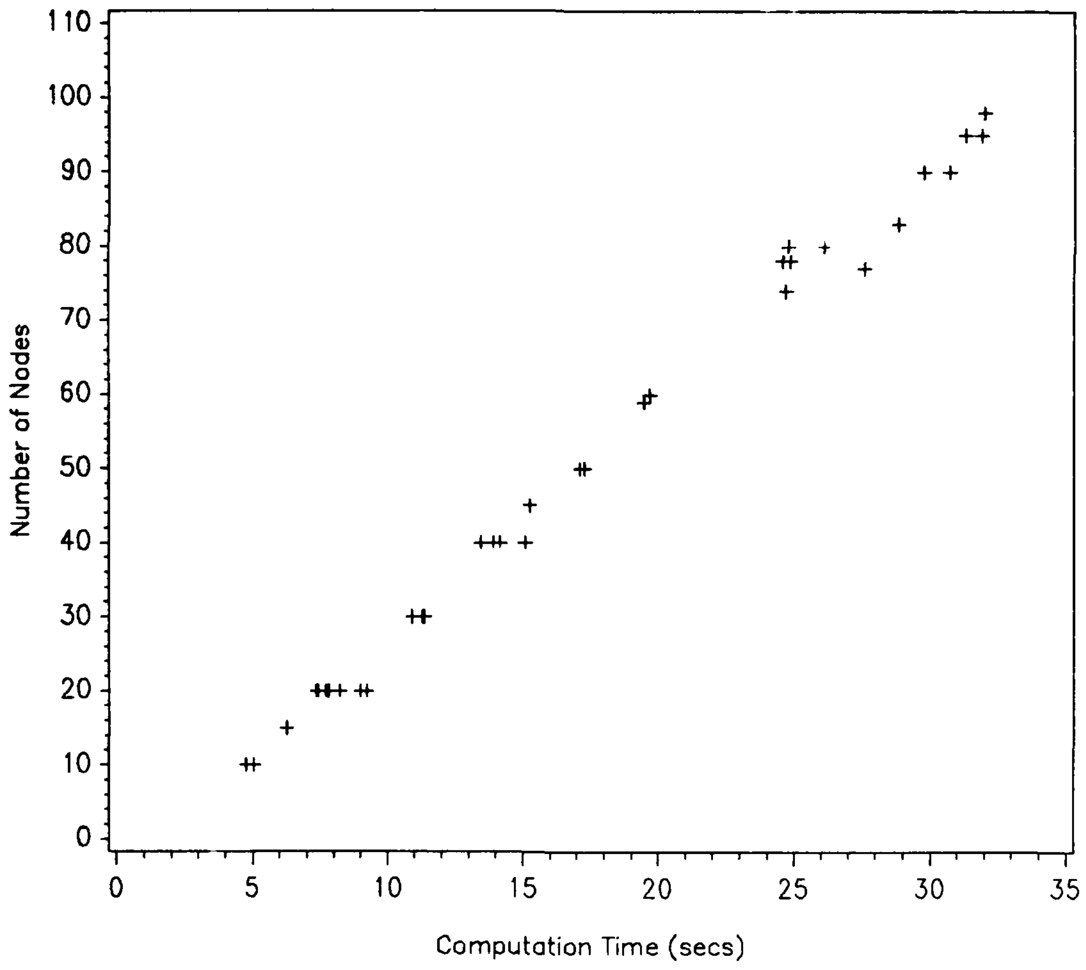
DTime = 1.83 +.312 * #Nodes R–Sq = 99.2

Two Shortest Paths Evaluated

Best Bound Time Performance

vs.

Number of Network Nodes



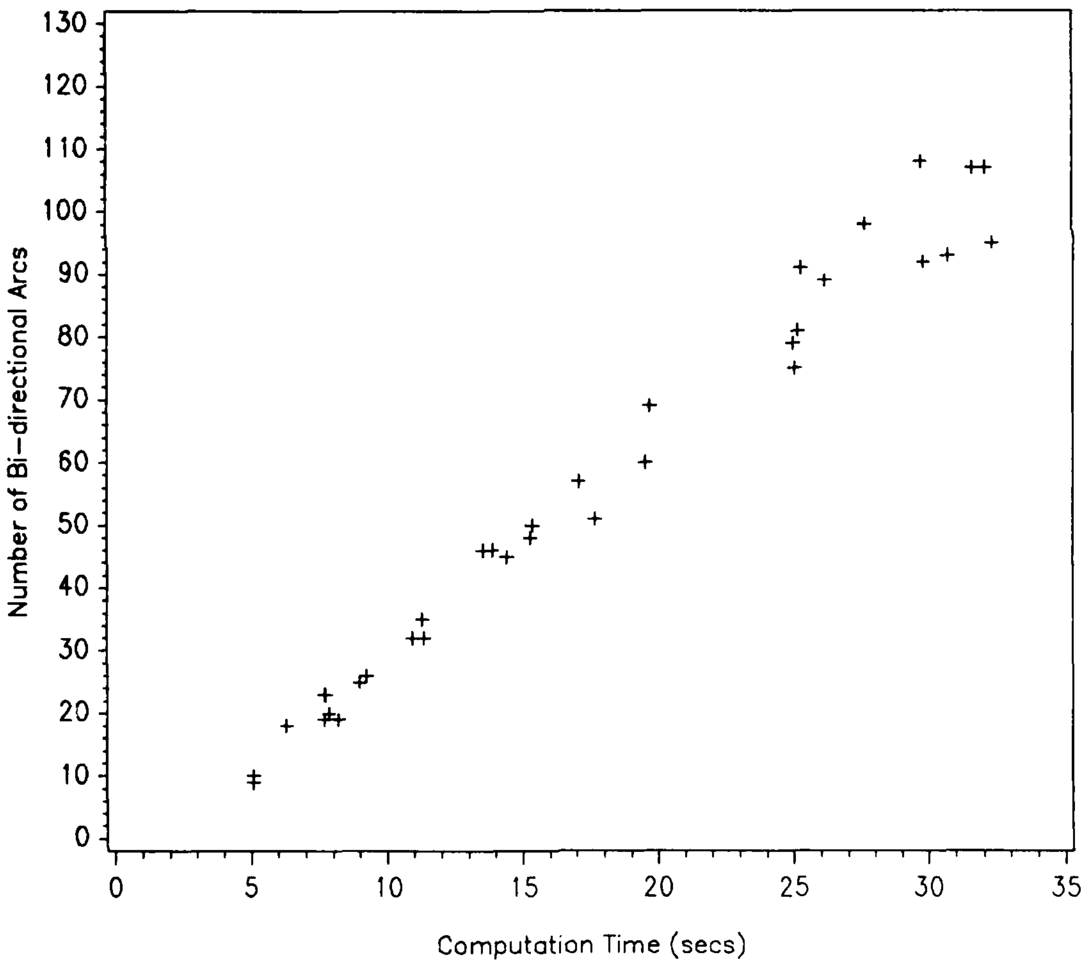
BBTime = 1.78 +.310 * #Nodes R–Sq = 99.3

Two Shortest Paths Evaluated

Depth First Time Performance

vs.

Number of Bi–directional Arcs



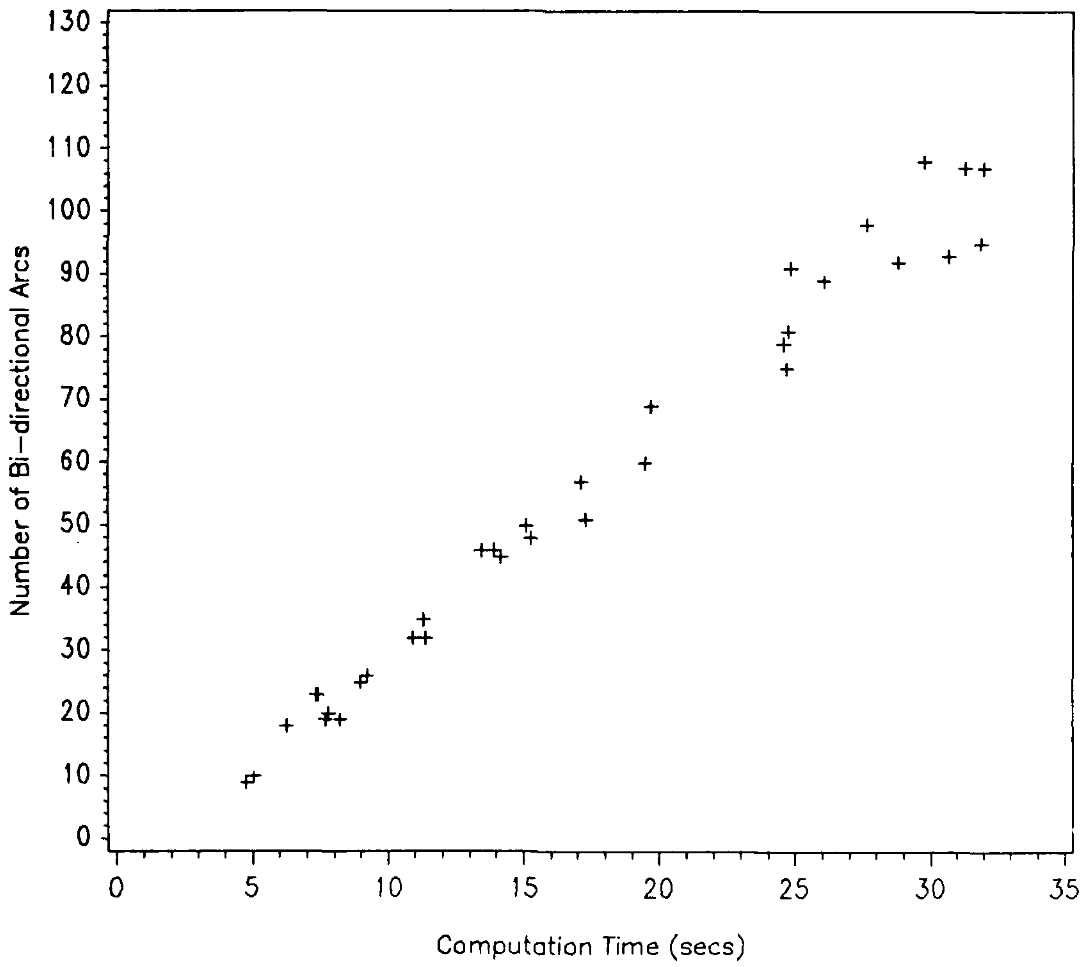
DTime = 1.94 +.282 * #BiArcs R–Sq = 97.9

Two Shortest Paths Evaluated

Best Bound Time Performance

vs.

Number of Bi-directional Arcs



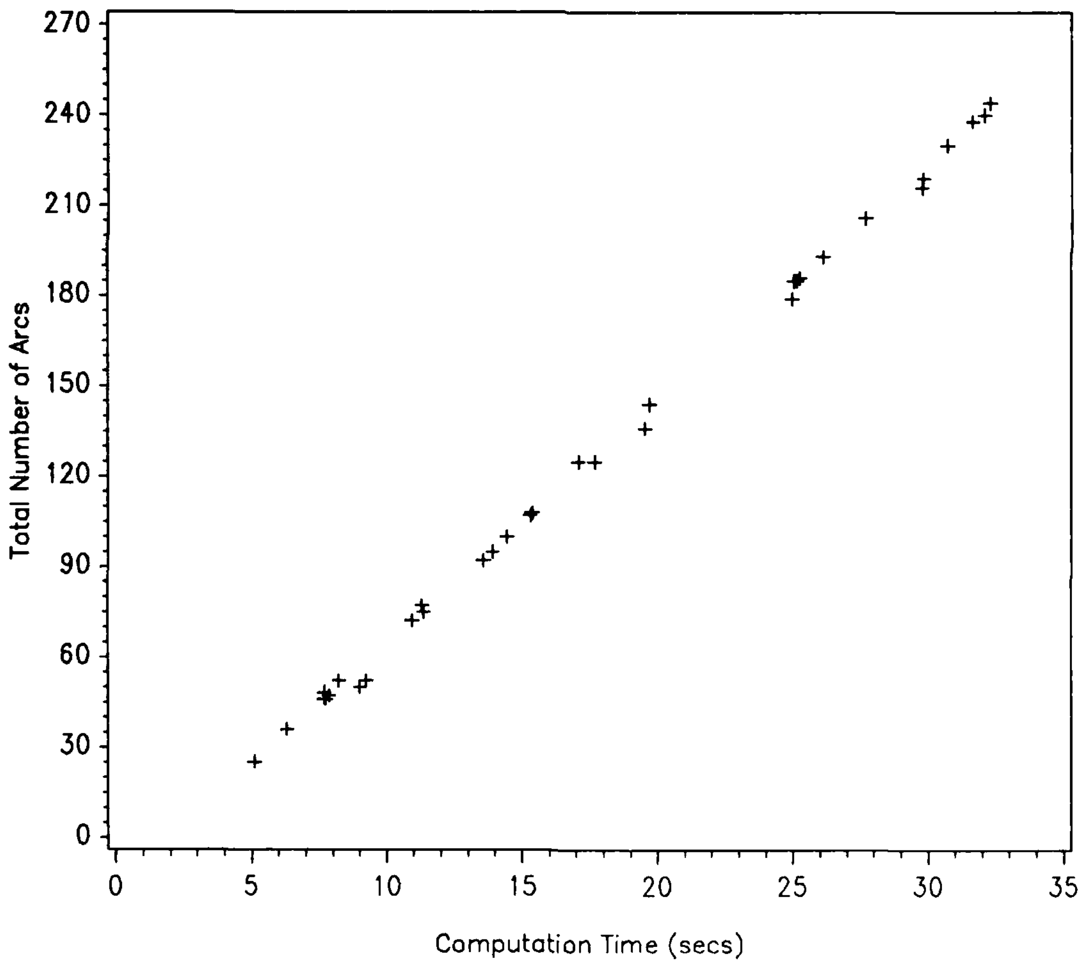
BBTime = 1.88 +.280 * #BiArcs   R-Sq = 98.1

Two Shortest Paths Evaluated

Depth First Time Performance

vs.

Total Number of Arcs



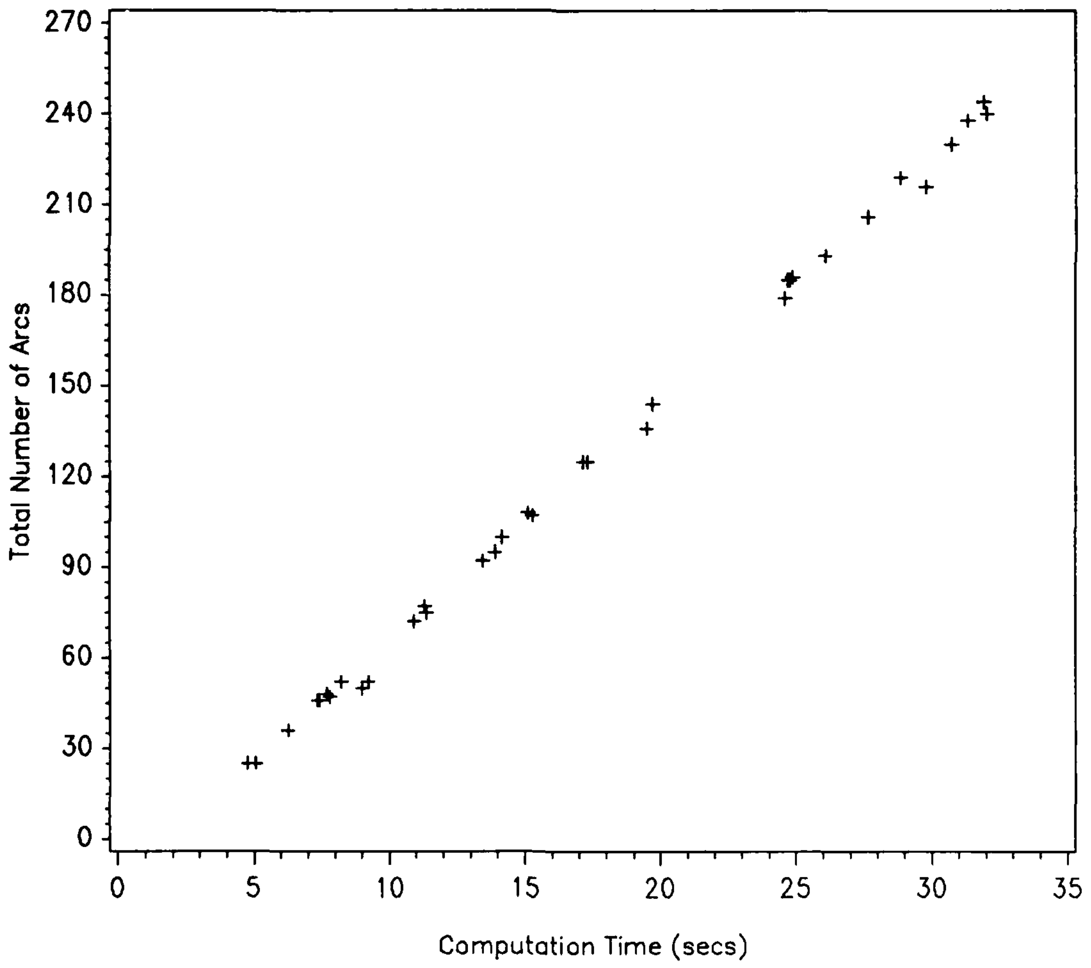
DTime = 1.98 +.125 * #TotArcs R–Sq = 99.9

Two Shortest Paths Evaluated

Best Bound Time Performance

vs.

Total Number of Arcs



BBTime = 1.93 +.125 * #TotArcs R−Sq = 99.8

VITA

Stephen Craig Daniels [redacted]. He graduated from Clinton Senior High School, Clinton, Missouri, May 1970. He entered the United States Air Force Academy (USAFA) and graduated in June 1974, receiving a Bachelor of Science in Mathematics.

After graduation he attended Undergraduate Pilot Training (UPT) at Moody Air Force Base (AFB), Georgia. He graduated from UPT in July 1975 and was assigned to Combat Crew Training School (CCTS), Castle AFB, California, where he learned to fly KC-135 aircraft. An outstanding graduate from CCTS, he was assigned to the 42nd Aerial Refueling Squadron. From 1976 to 1980 he was a Copilot, Aircraft Commander, and Squadron Executive Officer at Loring AFB, Maine. In 1980 the Air Force selected him to attend The Pennsylvania State University. He graduated in March 1982 with Master of Science degrees in computer science, industrial engineering, and operations research. From 1982 to 1985 he served as an Instructor and Assistant Professor in the USAFA Mathematics Department.

He published an article "Fuzzy Multi-criteria Integer Programming via Fuzzy Generalized Networks," in Fuzzy Sets and Systems, Vol 10, 1983.

He has given three presentations to professional organizations:

1. "Sensitivity Analysis in Linear Integer Programming and Applications to Goal Programming," presented to The Pennsylvania State Operations Research Colloquium, August 1982.

2. "Symmetry in and Computation of the Transient Output Process Solution for M/M/1 Queue," presented to the XXVI International TIMS Conference, Coppenhagen, Denmark, June 1984.

3. "Microcomputer Use in Teaching Math Modeling," presented to the Joint Service Mathematical Modeling Symposium, 1983 and 1984.

He belongs to the Operations Research Society of America and the Air Force Association.